Reference Manual

IBM 7070 Series Programming Systems

Autocoder

IBM Reference Manual

IBM 7070 Series Programming Systems Autocoder

Contents

Introduction

IBM 7070/7074 Autocoder is a symbolic programming system designed to simplify the preparation, correction, and interpretation of programs for the IBM 7070 and 7074 Data Processing Systems. This manual is a reference text and contains information to enable the programmer to use the Autocoder system.

Autocoder is a component of the 7070/7074 Compiler Systems Tape and forms an interlocking system with FORTRAN and the Report Program Generator. The Compiler Systems Tape may be obtained by sending a full reel of magnetic tape to the IBM Program Librarian. The compilers will be written on the tape which will be returned. It is suggested that a duplicate of the tape be made as soon as it is received so that one copy of the program can be kept in reserve. The extra copy should be used only in case the working copy becomes unusable. The reel of magnetic tape for the 7070/7074 Compiler Systems Tape should be sent to:

IBM 7070/7074 Program Librarian
International Business Machines Corporation
590 Madison Avenue
New York 22, New York

This manual assumes that the programmer is familiar with the methods of data handling and the functions of instructions in the 7070/7074 Data Processing System. This information is included in the IBM Reference Manual "7070 Data Processing System," form A22-7003-2.

The 7070/7074 Autocoder system is designed for use in installations which have a minimum of six 729 Model IV (or Model II) tape units and a machine with 5,000 words of core storage. This minimum configuration permits compilation of programs whose input is in the form of a card image tape and whose output is to be written on tapes for printing and/or punching off-line. The following additional equipment may be added to perform the indicated operations:

1. IBM *7500 Card Reader* is necessary, in addition to the six tape units, if *any* input is in the form of punched cards.
2. IBM *7550 Card Punch* is necessary if on-line punching of output is desired.
3. IBM *7400 Printer* is necessary if on-line printing of output is desired.
4. *As many as four additional tape units* may be used if the input is to be on more than one tape unit and/or if more than one program is to be compiled during a single machine run.

The specific machine requirements for each type of run which may be made using the Compiler Systems Tape are included in the 7070/7074 Data Processing System Bulletin "IBM 7070/7074 Compiler Systems: Operating Procedure," form J28-6105. Detailed operating instructions for each run, as well as the control cards necessary, are also included in this bulletin.

7070/7074 Autocoder is one of the powerful programming languages of today. Such languages have steadily evolved from languages requiring highly codified instructions closely related to the arithmetic capacities of the machine.

The first step in the evolution was the introduction of symbolic programming systems, such as 7070/7074 Basic Autocoder, in which a symbolic instruction is

written in place of each machine-language instruction. The programmer is thus able to code more easily and with greater meaning and the chance of errors is materially reduced.

The introduction of macro-instructions was a further step towards simplifying programming and reducing the time required to write a program. A macro-instruction is a symbolically-coded instruction which will produce a group of machine-language instructions. Two types of macro-instructions exist, substitution-type and generator-type.

The macro-instructions handled by 7070/7074 Four-Tape Autocoder are the substitution-type. The processor completes a "skeleton" routine by inserting parameters from the macro statement operand into this routine. The macro statement is then replaced by the completed routine. Four-Tape Autocoder also accepts symbolic machine instructions.

Autocoder, in addition to accepting symbolic machine instructions, handles generator-type macro-instructions. The task of compiling the proper sequence of instructions for the given macro statement is performed by the appropriate macro generator in the Library portion of the Compiler Systems Tape. In general, the operand of each macro-instruction must conform to a basic format. However, Autocoder also accepts certain macro-instructions with operands whose formats have not been pre-established "symbol by symbol." For example, the operand of ARITH or of LOGIC contains an "expression," the value of which is to be computed or the truth or falsity of which is to be determined. The macro generators scan the macro statements and compile the corresponding sequence of symbolic machine instructions; the operand need only conform to the rules of format which have been established and must not exceed certain specified lengths. Numerous illustrations of source program macro-instructions and their corresponding series of generated machine instructions are included with the macro-instructions under "Imperative Statements."

Macro-instructions to handle input/output operations are included in both the Autocoder and Four-Tape Autocoder systems. With certain restrictions, the source-program input/output statements are written the same for both Autocoders. However, the statements are handled differently by the processors of the two systems. In Four-Tape Autocoder, the processor substitutes the completed "skeleton" routine for the macro statement; in Autocoder, the macro-instruction is processed by means of a macro generator. A description of the input/output macro-instructions is included in this manual; a full discussion can be found in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," form J28-6033-1. Input/output operations are estimated to constitute an average of 40% of a program; the ability to handle these operations by means of macro-instructions represents a substantial gain in programming simplicity and efficiency.

The Autocoder language may be extended by adding new macro-instructions. Appropriate macro generators may be added to the system to process the newly-created macro-instructions. The necessary generator is written in the Autocoder language according to the instructions presented in the 7070 Data Processing System Bulletin "Additions to the IBM 7070 Autocoder; Writing Macro Generators for the IBM 7070 Autocoder," form J28-6053. The generator is then compiled and entered on a new Compiler Systems Tape and the corresponding macro-instruction may then be included in any program.

While macro-instructions provide the programmer with a set of powerful tools to solve problems without becoming enmeshed in the tedious details of analysis

and of storage assignment, the programmer may still exercise direct control over the minor details of his program, should it be necessary. Autocoder accepts and processes all symbolic machine instructions as well as macro-instructions; in fact, any program written in Basic Autocoder can be assembled without change by Autocoder. In addition, storage allocation can be specified by a set of control statements which have been provided.

In summary, the Autocoder system provides a number of advantages by introducing a powerful macro language. The programming and processing advantages are as follows:

1. Macro language eliminates the need for breaking down many frequently encountered tasks into a number of small steps by turning these tasks over to macro generators.
2. Full use is made of the symbolic programming devices of the system. The need for attention to the details of data flow, actual storage allocation, and decimal-point positioning is eliminated.
3. The programmer is allowed to write program steps that are meaningful in terms of the problem to be solved, rather than in terms of machine capacities.
4. The program may be easily broken into meaningful segments, allowing for greater flexibility and accuracy in reprogramming or recombining of program segments.
5. The program is made much more readable.
6. Programming speed is materially increased.
7. Programming is much easier to learn because fewer instructions must be written by the programmer and because the need for concern with many machine details is eliminated.
8. Macro language reduces programming errors by making use of tested macro generators rather than many individual machine instructions.
9. Errors in the input statement are detected by macro generators themselves; a message is issued indicating the location of the error and, generally, the type of error committed.
10. Programs are largely independent of individual machine characteristics and therefore are easier to transfer from one system to another.

FIGURE 1

IBM®

FORM X28-6417-2
PRINTED IN U.S.A.

Program ______________

Programmed by ______________

Date ________

7070 AUTOCODER CODING SHEET

Identification ______ (76–80)

Page No. ___ (1 2) of ______

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21 25 30 35 40 45 50 55 60) Basic Autocoder →	Autocoder → (65 70 75)
01				
02				
03				
04				
05				
06				
07				
08				
09				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				

Coding Sheet

A programmer, coding a program for the 7070 or 7074 in Autocoder, writes all Autocoder statements on the 7070 Autocoder Coding Sheet, form X28-6417-2 (see Figure 1). The coding sheet indicates by column numbers the input card format for both the Basic Autocoder and the Autocoder Systems. Each line of the coding sheet is punched into the indicated columns of a corresponding IBM 7070 Autocoder Input Card, electro A18265 (see Figure 2). An explanation of the purpose of each heading on the coding sheet is given below.

Heading Line

The heading line consists of the spaces labeled "Program," "Programmed By," and "Date." The information entered in these spaces is for identification of the program and is not to be punched into input cards.

Page Number (Columns 1-2)

A two-character page number sequences the coding sheets. Any alphameric characters may be used, providing they can be read into or can be translated on output by the input/output equipment of the 7070 or 7074 system used to process the Autocoder source program. (This applies in general to the usage of alphameric characters in all Autocoder statements.) The standard collating sequence should be followed in sequencing the pages. Alphameric characters which are not acceptable to various input/output equipment and the collating sequence may be found on page 9 of the IBM Reference Manual "7070 Data Processing System," form A22-7003-2.

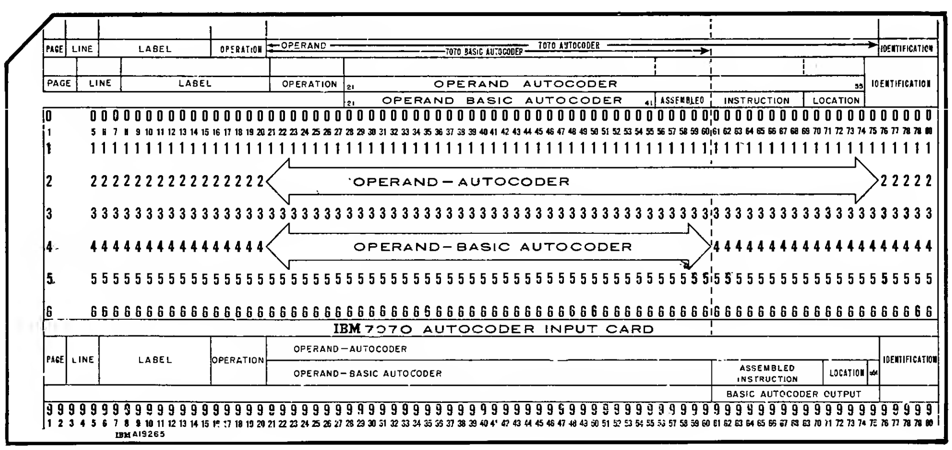

FIGURE 2

Line (Columns 3-5)

The first twenty-five lines on each sheet are prenumbered 01 through 25. Also provided are five unnumbered lines at the bottom of the sheet that may be used for additional lines or for inserts. Since provision is made for a three-character line number for the sequencing of the coding entries on the sheet, inserts may be readily made through the use of the optional third character. For example, inserts between lines 10 and 11 may be made by writing 101, 102 and 103 or 10A, 10B, and 10C, etc. in columns 3-5 of the unnumbered lines and placing the resulting cards after the card for line 10. Any alphameric characters may be used for all three characters of the line number so long as the standard collating sequence is followed in sequencing the lines.

The sequence of the cards entered into the processor will be checked by the page and line numbers punched in the source program deck. Any variation from the collating sequence will be noted in the warning and error message area of the program listing produced during a compilation. However, source-language input will be compiled in the order encountered.

Label (Columns 6-15)

The label column is used to represent the location of data or instructions in the machine. Only instructions or data which will be referred to elsewhere in the program need have a label. In all other cases, the label column is blank. A label may be a symbolic location or an actual address. Each symbolic label must be unique, i.e., it may *not* appear more than once in the label column.

Operation (Columns 16-20)

The operation column contains either a macro statement or the mnemonic representation of the machine operation to be performed. In certain cases, the column may be left blank. Actual machine operation codes are never used. The Autocoder symbolic machine instructions and macro-instructions are composed of from one to five alphameric characters, and are written left-justified in the operation column. Operation codes are categorized as "declarative," "control," and "imperative" statements. A description of these three types of statements is contained on pages 20, 87 and 103, respectively. If an invalid operation is used, a NOP will be generated.

Operand (Columns 21-75)

The operand contains the actual or symbolic address of the information which is to be acted on by a particular command, or other parameters to be utilized by a macro generator. The operand may contain 55 columns of information as input to the Autocoder processor as opposed to the 40 columns for the Basic Autocoder processor. When field definition, address adjustment, or indexing is used in conjunction with the address, it is included in the operand. The operand may contain the actual data to be operated on by an instruction, referred to as a "literal." It may also be used to specify index words, electronic switches, channels, units, channel and unit, arm and file, inquiry and unit record synchronizers, latch numbers, and alteration switches.

Identification (Columns 76-80)

Program identification is punched into columns 76-80 of all cards in the source program deck. The identification which appears on the *first* card of the source program deck will be punched in the identification field of each card of the object program deck and printed on each page of the output program listing by the Autocoder processor. A means is thus provided for relating all output to the proper symbolic source-language input.

Any alphameric characters acceptable to the 7070/7074 input/output equipment may be used in the identification entry. Alphabetic and special characters will print as such on the program listing. However, only the second digit of each double-digit representation of these characters will be punched in the identification field of the condensed cards of the object program decks since these cards must be numerical. For example, "A" will be punched as "1," "*" will be punched as "6," etc. If the first card of the source program deck has a blank identification field, the identification on the program listing will be blank and zeros will be punched in the cards of the object program deck, the double-digit representation of a blank being 00.

Remarks and Comments

Remarks and comments may be included for description. They will appear in the symbolic output but will produce no entry in the object program deck and consequently will not affect the operation of the program.

Remarks may be included anywhere in the operand, provided they are separated by at least two blank spaces from the operand of the instruction. As noted on page 60, an @ symbol may not appear in remarks which are on the same line as an alphameric constant. Otherwise, remarks may include any acceptable characters.

Comments cards allow the programmer to insert complete lines of descriptive information in the program. A comments cards is identified by placing an asterisk in column 6 of the label column. Any part of the label, operation, or operand columns may be used for the description. Comments cards are useful as descriptive headings for various sections of a program, such as operating instructions, or where the operand column of an instruction does not allow enough room for necessary remarks.

Remarks and comments may be used in a program as follows

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 Basi 50
01	*NET PAY C	ALCUL	ATION
02		ZA1	GROSS PUT GROSS PAY IN ACC 1
03		S1	TAX DEDUCT INCOME TAX
04		S1	FICA DEDUCT FICA
05			

Parameters

Address Types

The following types of addresses may appear in the label and/or operand fields of Autocoder statements: blank, actual, symbolic and literal. A description of these address types and the rules governing their usage follow.

Blank

The label column may be blank if the corresponding entry is not referred to elsewhere in the program. A blank operand is valid for certain machine operations (see Appendix D) and in the following control and declarative statements:

1. ORIGIN Control
2. LITORIGIN Control
3. END Control
4. XRESERVE Control
5. SRESERVE Control
6. XRELEASE Control
7. SRELEASE Control
8. DC (Header Line)
9. DLINE (Header Line)

The effect of a blank operand varies for each of the control and declarative statements in which it may appear. These are explained fully under the discussion of the respective statements. If a Priority Release command has a blank operand, the processor will insert 0097. In all other machine operations where a blank operand is valid, the processor will insert 0000. If a blank operand is invalid, an error message will be produced.

Actual

An actual address may be from one to four digits, written left-justified on the coding sheet. Leading zeros may be omitted. An actual address is valid only in the label and operand of symbolic machine instructions and in the operand of certain macro-instructions and control statements.

In certain instructions, an actual operand may only assume values within a limited range, e.g., a value of 1 through 4 in an alteration switch instruction and a value of 1 to 30 in an electronic switch instruction.

An actual address in the label column of a symbolic machine instruction will cause the instruction to be assigned that actual location. The contents of the location assignment counter being used will not be changed and the actual location, with the exception of index word addresses, will not be reserved. Hence, the programmer should be extremely cautious when using an actual label.

The following example illustrates the use of actual addresses in symbolic machine instructions:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		NOP	32
02		.	
03	0	B	0308
04		.	
05		HP	1
06			

Symbolic

A symbolic address is valid in the operand of most statements and in the label column of all but the control statements. A symbolic address may contain from

one to ten characters with the following restrictions: the first, or leftmost, position must be a letter; the remaining characters may be letters or numbers (no special characters); blanks may not appear within the symbolic address. A symbolic address placed in the label column of a declarative or imperative statement is automatically associated with an actual storage location assigned by the processor's location counter. Further reference to a certain instruction may be made or operations on a particular data field may be performed by writing the symbolic name assigned to the instruction or data field in the operand of an imperative, declarative, or control statement.

Consider the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ROUTINE	ZA1	FIELD
02		.	
03		.	
04		BH	ROUTINE
05			

If FIELD is a one-word field assigned to location 3000, the first entry above will result in the assembled instruction +1300093000. If the assignment counter had been at 5000 when the first entry was encountered, the label ROUTINE would have been associated with location 5000. Thus the entry in line 04 above would result in the assembled instruction –4000095000.

The asterisk, *, is a special symbol which is valid in the operand only. If an actual address has not been written in the label column, the processor will assign the location of the instruction being processed, contained in the location assignment counter, to *. For example, if the instruction

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		ZA1	*
02			

has been assigned to location 4440, then the * in the operand of that instruction will also be assigned location 4440; i.e., the assembled machine instruction will be +1300094440.

If an actual label *has* been entered in the label column and an * in the operand, the processor will assign the actual address to *. Thus, the instruction

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	324	ZA1	*
02			

would result in the assembled machine instruction +1300090324.

Use of the asterisk address will reduce the number of symbols required in the label column. Unless there is a note to the contrary, the special symbol, *, may be used as an operand address wherever this manual indicates that a symbolic address is valid.

Literal

A literal is the actual data to be operated on by an instruction. The literal is valid only in the operand of an imperative statement, and its appearance causes the processor to assign a storage location to the literal. In order to conserve storage, literals are packed into words. The processor assigns field definers to the literal and incorporates the storage address and field definers into the instruction being processed. Once a literal has been stored, it will be re-used each time it is referenced again, except when the Litorigin Control statement is used (see page 88). Address adjustment and indexing (both discussed in later sections) should *not* be appended to literals. Literals should *not* be used for temporary storage, i.e., the operands of Store instructions should not be literals.

Certain imperative instructions (e.g., Priority Control, Index Word Load, Tape Write) operate on full words and do not permit field control. If a literal which is less than ten digits, or an adcon, is used as the operand of any of these instructions, it will be converted to ten-digit form by being right-justified in a word. Thus, if −50 is entered as the address portion of an Index Word Load instruction (XL), the actual constant stored will be −0000000050. Also, assume that ATABLE is the label of an instruction or data occupying location 2000. If the adcon −ATABLE is entered as the address portion of an Index Word Load and Interchange command (XLIN), the representation will be −0000002000, and, at object time, the contents of the specified index word will be replaced by −0020000000. In general, however, a literal should be written in the exact form in which it is to be used with leading zeros included where necessary.

Four principle classes of literals are permissible: automatic-decimal numbers, floating-decimal numbers, address constants (adcons), and alphameric constants.

Automatic-Decimal Numbers

A literal having either of the following sets of characteristics is included in the automatic decimal class.

1. A signed number, one to *twenty* digits in length, which is referred to by a *macro-instruction.* A decimal point *may be* included in the literal.
2. A signed number, one to *ten* digits in length, which is referred to by a *symbolic machine instruction.* A decimal point *may not be* included in the literal.

An automatic-decimal number, referred to by a macro-instruction and described by the first set of characteristics above, will be examined by the Autocoder processor for decimal-point inclusion. A decimal point indicates the magnitude of the number according to ordinary usage and the desired decimal-point placement; it is neither stored in the constant nor saved with it. Using Autocoder macro generators, the processor generates instructions for shifting, decimal alignment, bridging words, and handling field definers. If the decimal point falls to the right of the rightmost digit of the number, it may be omitted and the number will be considered an integer. The following are examples of automatic-decimal literals which might appear in the operand of a macro-instruction:

```
+1234567890123.4567890
-.12345
-1.23
+1
```

An automatic-decimal number described by the second set of characteristics above is actually a subset of one described by the first set of characteristics. Because the number is referred to by a symbolic machine instruction, the literal is restricted to ten digits in length and all instructions for shifting and decimal alignment must be written by the programmer.

An automatic-decimal number may also be defined by a DC subsequent entry (see page 55).

Floating-Decimal Numbers

Floating-decimal numbers are permitted as a literal entry in macro-instructions only. A number may be expressed in the form

$$(\pm n)\ (10^{\pm m})$$

where n is an integer or decimal number of not more than eight digits and m is a one-digit or two-digit exponent. Floating-decimal numbers are related to this form and are. entered according to the format ±nF±m. Thus, the number $-.12345678 \times 10^3$ would be represented by −.12345678F+3. If the sign preceding m is omitted, m is considered to be positive. The exponent m may be omitted if equal to 0, provided ±nF is not followed by another literal entry. The Autocoder processor will consider the signs, the value of n, and the value of m, and generate a standard 7070 floating-decimal word. The following are examples of floating-decimal numbers which might appear in the operand of a macro-instruction:

−3.4F
+34.567893F−2
−31.92F+7
−29567.1F−3
+12546F+15

Additional examples of floating-decimal numbers may be found in the IBM Reference Manual "7070 Data Processing System," form A22-7003-2.

A floating-decimal number may also be defined by a DC subsequent entry as described on page 57.

Address Constants (Adcons)

An adcon is a special-purpose numerical literal used to produce a four-digit constant whose value is the actual address assigned to a symbolic address. The address constant is treated in the same manner as a numerical literal. An adcon is entered in the form ±SYMBOL. The following example illustrates the use of an adcon:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21 25 30 35 40 45)
01	ANYLABEL	S	FIELD
02		.	
03		.	
04		ZA1	+ANYLABEL
05			

If the symbolic address, ANYLABEL, is assigned the actual location, 2000, the use of the adcon, +ANYLABEL, in the operand of line 04 will cause the processor to produce a constant of +2000. If the address constant, +2000, is assigned to location 4576 with field definers (6, 9), the entry on line 04 would result in the assembled instruction +1300694576. The execution of this instruction at object program time would place the number, +0000002000, the actual address assigned to ANYLABEL, in accumulator 1.

An adcon may also be defined by a DC subsequent entry (see page 58).

Alphameric Literal

An alphameric literal consists of alphameric characters preceded and followed by the @ character. A literal in the operand of a symbolic machine instruc-

tion may not be more than five characters in length and, in the operand of a macro-instruction, may not be more than 120 characters in length. All characters between the initial @ character and the second @ will be converted to double-digit form and assigned to core storage locations. The sign of each word used to contain the characters will be alpha. If the constant produced by a literal is more than one word in length, but *not* a multiple of five alphameric characters, the double-digit representation of alphameric blanks will be generated in the unused low-order positions of the last word of the constant, i.e., the constant will *not* be packed.

Alphameric literals may *not* include the @ character. Alphameric literals may, however, contain any other character (except for the record mark) which may be read by the input device used.

The following are examples of alphameric literals which might appear in the operand of macro-instructions or symbolic machine instructions:

@ABCD@
@A100X@
@12345@
@.⌑+$*@
@−/,%#@

The following literal would be valid in the operand of a macro-instruction only:

@THIS LITERAL IS LONGER THAN 5 CHARACTERS@

Alphameric constants, including those containing the @ character, may be defined by a DC subsequent entry (see page 60).

Field Definition

Field definition may be written immediately following the operand address of symbolic machine instructions. The field definers are the digit positions of a ten-digit word numbered 0 to 9. The format for entering field definers is as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50 55 Basic Autocoder
01			ADDRESS(STARTPOSITION,ENDPOSITION)
02			

The starting and ending positions are the field definers. When operating on a single digit, the comma and ending digit position may be omitted. Field definers may be omitted entirely when operating on a whole word or a field defined by a declarative statement (see pages 22 through 81).

Field definition may not be used in the operand of a macro-instruction. However, both macro-instructions and symbolic machine instructions may refer to the label of a Define Area (DA), Define Constant (DC), or Define Line (DLINE) subsequent entry, each of which may specify field definition.

Autocoder symbolic machine operation codes which permit field definition to be associated with the address are indicated in Appendix D. Field definition

may be used with both actual and symbolic addresses as illustrated in the following examples:

Line	Label	Operation	OPERAND
01		S1	*(8,9)
02		A1	MANNO(0,2)
03		ST2	1024(2,8)
04		ZA1	ALPHA(5)
05		C1	MANNO(0,9)
06		AS1	1024(0,9)
07		ZA1	-SYMBOL(2,3)
08			

In lines 05 and 06 above, field definers (0, 9) could have been omitted since the instructions will be operating on the entire word.

When used with literals, field definition will be relative to the literal itself. Thus, the adcon −SYMBOL(2, 3) refers to the third and fourth digits of the location assigned to SYMBOL. If location 9876 has been assigned to SYMBOL, −SYMBOL (2, 3) would be equivalent to −76. (See "Relative Field Definition" on page 41.)

Address Adjustment

Address adjustment allows the programmer to refer to an entry which is a given number of locations preceding or following a symbolic address. Address adjustment is permitted with adcons and with all symbolic addresses, except the single-address operand of a DRDW statement (see page 78). It should not be used with actual addresses or other literals.

With *symbolic machine instructions,* address adjustment is indicated by writing a plus or minus sign followed by one to four digits immediately after the symbolic address and following any field definers.

Address adjustment with symbolic machine instructions is written as follows:

Line	Label	Operation	OPERAND
01			MANNO+5
02			MANNO-2
03			MANNO(0,2)+5
04			*-1
05			*(0,4)+10
06			

If location 2150 has been assigned to the symbolic address MANNO, then 2155 will be assigned to MANNO+5, and 2148 will be assigned to MANNO−2. The entry MANNO (0, 2)+5 refers to the first three digit positions of location 2155. Similarly, if 2196 is the location of an instruction containing *−1 as an operand, then location 2195 will be assigned to this operand.

With *macro-instructions,* address adjustment is indicated by writing a plus or minus sign followed by one to four digits after the symbolic address. The sign

and digits must be enclosed by parentheses. The left parenthesis must be in the column immediately following the last character of the symbolic address being modified, except when the address adjustment goes to a continuation card (see "Continuation Cards").

Address adjustment with macro-instructions is written as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01			LIST(+3)
02			LIST(−10)
03			

If location 1560 has been assigned to the symbolic address LIST, then 1563 will be assigned to LIST (+3), and 1550 will be assigned to LIST (−10).

Address adjustment of an *adcon* will cause a special function. The value of the address will be modified before the constant is created. Address adjustment of an adcon in a symbolic machine instruction is written as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	ZA1	+ANYLABEL+1
02			

If ANYLABEL is assigned to location 2000, the above entry will cause the creation of a literal (or adcon) value of 2001 to be stored for reference. In a macro-instruction, the format in the operand would be +ANYLABEL(+1).

The programmer should be careful when using address adjustment since it may become a source of error when a program is modified. For example, inserts and deletions of program entries could change addresses in such a way that *+10 should now be *+9. It should also be noted that since it is not known how many machine instructions will be generated in place of a macro-instruction, address adjustment must not be used on a symbolic label in amounts that would carry the address into or across a macro-instruction.

Index Words

Indexing

The use of an index word in an instruction for the purpose of indexing will cause the indexing portion of the index word to be added algebraically to the address portion of the instruction and this new address is used for the operation. Indexing of symbolic and actual addresses may be specified in the operand of all imperative statements and Branch Control and End Control statements. Literals should not be indexed.

With the *Branch Control and End Control statements* and with *symbolic machine instructions,* the address of an index word follows the operand address, after field definers and address adjustment, if any, and is always preceded by a plus sign. An index word may be written symbolically or as the actual one- or two-digit number (1-99). When a symbolic name is used, the processor will

automatically assign an actual index word address. When the actual number is used, the format is Xn. The X indicates that an index word address rather than address adjustment follows the plus sign. The n is the actual one- or two-digit number of the word. When used for other than indexing purposes, the form Xn will be considered a symbol. Indexing with Branch Control and End Control statements and symbolic machine instructions is written as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01			MANNO+X2
02			MANNO−15+X2
03			MANNO(0,4)+15+LOOP
04			2344+X2
05			2344+LOOP
06			2344(0,5)+LOOP
07			

Autocoder will interpret +X2 as index word 2 and +LOOP as the symbolic designation of an index word. Note that blanks are not permitted within the address modification.

With *macro-instructions,* the address of an index word follows the operand address and is enclosed by parentheses. The left parenthesis must be in the column immediately following the last character of the address being modified, except when the indexing goes to a continuation card (see "Continuation Cards"). Actual index word numbers are not preceded by an X and the actual or symbolic index word is not signed. In macro-instructions, the absence of a plus or minus sign preceding an index word distinguishes the index word from a signed, one- or two-digit address adjustment. If both address adjustment and indexing are used, one set of parentheses must enclose them both and the indexing must *precede* the address adjustment. Indexing with macro-instructions is written as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01			TABLE(ROW)
02			LIST(34)
03			TABLE(ROW−10)
04			LIST(34+17)
05			

Autocoder will interpret 34 as index word 34 and ROW as the symbolic designation of an index word. Note again that blanks are not permitted within the address modification and, also, that field definition may not be used with macro-instructions.

Uses Other Than Indexing

Index words may be specified in symbolic machine instructions as the first entry in the operand of index word commands such as Index Word Load (XL) and Index Word Load and Interchange (XLIN). They may also be used in commands such as Record Gather (RG) and Record Scatter (RS). The specification of an *index word* in a symbolic machine instruction does not prevent the use of *indexing.* The index word may be written in actual or symbolic form. When written in actual form, however, only the one- or two-digit address of the index word should be written; Xn would be interpreted as the symbolic designation of an index word.

The following are examples of the use of index words for other than indexing purposes:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
01		XL	2,,CONSTANT
02		XL	X3,,+0000010100
03		RG	BASE,,SCATAREA
04			

BASE and X3 will be interpreted as the symbolic designation of an index word and 2 as index word 2.

Electronic Switches

Electronic switches may be referred to by a symbolic name or by their one- or two-digit actual number (1-30). Symbolic names may be assigned to an actual switch number by use of the declarative statement EQU (see page 82). As explained later, symbolic references to electronic switches will be assigned to an actual address during compilation.

Unlike index words (see page 18), electronic switches will *not* be reserved if the location assigned to an imperative statement or if any location defined by a declarative statement falls within the range 0101-0103.

Instructions referring to electronic switches are written as follows:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
01		BES	19,,COMPUTE
02		BSF	END,,LOOP
03		ESN	28
04			

END will be interpreted as the symbolic designation of an electronic switch and 19 and 28 as electronic switches 19 and 28.

Input/Output Unit and Alteration Switch Designations

The following items may be specified in actual or symbolic form in the operands of those instructions which refer to the particular items: channel, unit, combined channel and unit, combined arm and file, unit record synchronizers, inquiry synchronizers, and alteration switches. The declarative operation EQU is used to equate symbolic names to item numbers (see page 85).

Continuation Cards

Certain Autocoder statements make provision for more parameters than may be contained in the operand (columns 21-75) of a single line on the Autocoder coding sheet. When this is the case, the appropriate section of this manual will indicate that "Continuation Cards" may be used. Thus, when specifically permitted, the operand of a given line on the Autocoder coding sheet may be continued in the operand of from one to four additional lines which immediately follow.

The label and operation columns must be blank and the continuation of the operand must begin in column 21; i.e., it must be left-justified in the operand column of the coding sheet. The operand need not extend across the entire operand column of either the header card or continuation cards but may end with the comma following any parameter. Remarks may appear to the right of the last parameter on each card provided they are separated from the operand by at least two blank spaces.

Illustration of the use of continuation cards are included throughout the examples illustrating the various statements.

If a continuation card follows a statement that does not permit continuation cards, the compiler will generate a NOP and issue an error message. Additional restrictions regarding the use of continuation cards with macro-instructions appear on page 106.

Reservation of Index Words and Electronic Switches

The assignment of actual addresses to symbolic index word and electronic switch names occurs in Phase III of the Autocoder processor. The initial availability of index words and electronic switches is determined by a table which is included in the Compiler Systems Tape. This table originally indicates that index words 1 through 96 and electronic switches 1 through 30 are available for assignment to symbolic references; index words 97 through 99 are not available. The initial setting of this table may be altered, however, as described in the 7070/7074 Data Processing System Bulletin "IBM 7070/7074 Compiler System: Operating Procedure," form J28-6105.

During the first pass of Phase III, references to the *actual* addresses of index words and electronic switches are collected and the availability table is updated. At the end of this pass, the table indicates which index words and electronic switches are not available for assignment to symbolic references.

Both index words and electronic switches may have been made unavailable before the start of assignment in one of the following ways:

1. The initial setting of the availability table indicated that the index word or electronic switch was not available for assignment.
2. The one- two-digit number of the index word or electronic switch was used in the operand of a symbolic machine instruction to specify indexing or as a parameter which is *always* an index word or electronic switch, e.g.,

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		BLX	5,LOCATION
02		ESN	16
03			

3. The one- or two-digit number of the index word or electronic switch was used in the operand of an EQU statement, e.g.,

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	NAME	EQU	3,X
02			

When the index words or electronic switches are reserved because of actual usage in the statements described above, the position or order of the statements within the program is not important; any such reference will make the index word or electronic switch unavailable at the end of this pass.

During the assignment pass of Phase III, index words and electronic switches are reserved as they are encountered during assignment. Index words and electronic switches may be reserved in the following ways. The first two methods apply to both index words and electronic switches; the third applies only to index words.

1. During the assignment pass, each instruction is examined for reference to the symbolic name of an index word or electronic switch. When such a reference is found, an actual address is assigned and the availability table is changed so that the assigned index word or switch is no longer available for later assignment.

2. If the one- or two-digit address of an index word or electronic switch is used or is included in the operand of an XRESERVE or SRESERVE statement (see page 99), the corresponding index word or electronic switch is reserved.

3. If a statement has been assigned an address in the index word area

   a. by means of an actual label or

   b. by means of an ORIGIN statement which refers to an *actual address*

   the corresponding index word will be reserved. These entries should normally appear at the beginning of the program or immediately following each LITORIGIN statement. Otherwise, symbolic names may have previously been assigned to these same index words. (This method does not apply to electronic switches.)

The preceding methods allow efficient use of index words and electronic switches during a sectionalized or multi-phase program, particularly when used in conjunction with the LITORIGIN statement. Extreme caution should be used, however, to avoid the conflicting usage of an index word or electronic switch which may result from the assignment of more than one name or function to the same address.

If the symbolic name or actual address of an index word or electronic switch appears or is included in the operand of an XRELEASE or SRELEASE statement (see page 101), the specified index word or electronic switch will again be made available, regardless of the method by which it was reserved. It will not, however, be used for symbolic assignment until all other index words or electronic switches have been assigned for the first time.

If, at any time during the assignment pass, the compiler finds that there are no more index words available for assignment, the warning message "NO MORE INDEX WORDS AVAILABLE" will be placed in the object program listing, the table will be altered to show that index words 1 through 96 are available, and the assignment will continue as before. If the compiler finds that there are no more electronic switches available for assignment, the warning message "NO MORE ELECTRONIC SWITCHES AVAILABLE" will be placed in the object program listing, the table will be altered to show that electronic switches 1 through 30 are available, and assignment will continue as before. The resultant conflicting usage of index words or electronic switches may be avoided by reducing the number of symbolic names used, e.g., through the proper use of the EQU, XRELEASE, or SRELEASE statements.

As noted in Appendix C, index words 97 through 99 are *never* available for assignment to symbolic names by the compiler; also, index words 93 through 96 may have been made unavailable for assignment.

Declarative Statements

Autocoder declarative statements provide the processor with the necessary information to complete the imperative operations properly. Declarative statements are never executed in the object program and should be separated from the program instruction area, placed preferably at its beginning or end. Otherwise, special care must be taken to branch around them so that the program will not attempt to execute something in a data area as an instruction. If the compiler does encounter such statements, a warning message will be issued. 7070/7074 Autocoder includes the following declarative statements: DA (Define Area), DC (Define Constant), DRDW (Define Record Definition Word), DSW (Define Switch), DLINE (Define Line), EQU (Equate), CODE, DTF (Define Tape File), DIOCS (Define Input/Output Control System), and DUF (Descriptive Entry for Unit Records). DA, DC, DTF, and DLINE require more than one entry.

The DA statement is used to name and define the positions and length of fields within an area. The DC statement is used to name and enter constants into the object program. Since the 7070 and 7074 make use of record definition words (RDWs) to read, write, move, and otherwise examine blocks of storage, the DA and DC statements provide the option of generating RDWs automatically. When so instructed, Autocoder will generate one or more RDWs and assign them successive locations immediately preceding the area(s) with which they are to be associated. An RDW will be of the form ±00xxxxyyyy, where xxxx is the starting location of the area and yyyy is its ending location. These addresses are calculated automatically by the processor.

In some cases, it may be more advantageous to assign locations to RDWs associated with DA and DC areas in some other part of storage, i.e., not immediately preceding the DA or DC areas. The DRDW statement may be used for this purpose. The DRDW statement may also be used to generate an RDW defining *any* area specified by the programmer.

As many as ten digital switches may be named and provided by the DSW statement for consideration by the SETSW and LOGIC macro-instructions. Each switch occupies one digit position in a word, can be set ON or OFF, and is considered as logically equivalent to an electronic switch. It cannot, however, be referred to by electronic switch commands, e.g., ESN, BSN, etc. An individual switch or the entire set of switches in a word may be tested or altered as desired.

Through use of the DLINE statement, a means is provided for specifying both the editing of fields to be inserted in a print line area and the layout of the area itself. The area may include constant information supplied by the programmer. The area may also be provided with additional data during the running of the object program by means of EDMOV or MOVE macro-instructions.

The declarative statement EQU permits the programmer to equate symbolic names to actual index words, electronic switches, arm and file numbers, tape channel and unit numbers, alteration switches, etc., and to equate a symbol to another symbol or to an actual address.

The DIOCS, DTF, and DUF statements are used when required by the Input/Output Control System. DIOCS is used to select the major methods of processing to be used, and to name the index words used by IOCS. Each tape file must be described

by Tape File Specifications, produced by DTFS. In addition to information related to the file and its records, the File Specifications contain subroutine locations and the location of tape label information. A DUF entry must be supplied for every unit record file describing the type of file and the unit record equipment to be used. The DUF also supplies the locations of subroutines written by the user that are unique to the file.

A full description of the DIOCS, DTF, and DUF statements is contained in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," form J28-6033-1. Brief descriptions of these three declarative statements and detailed descriptions of the formats and functions of each of the other 7070/7074 Autocoder declarative statements follow below.

DIOCS—Define Input/Output Control System

When the Input/Output Control System is to be used in a program, a DIOCS statement must be used to select the major methods of processing to be used. This statement also allows the naming of the index words used by IOCS.

Source Program Format

The basic format of the DIOCS statement is as follows:

Line	Label	Operation	OPERAND (Basic Autocoder → 21–60)	Autocoder (61–72)
01	ANYLABEL	DIOCS	IOCSIXF,IOCSIXG,IOCSIXH,CHANn,OPENn,EORn	,CHPT,IGENn
02				

ANYLABEL is any symbolic label; it may be omitted. The entry DIOCS must be written exactly as shown.

The first item in the operand, IOCSIXF, is used to specify the first IOCS index word for programs using tape files. This item may be a symbolic name or an actual one-digit or two-digit index word address in the range 3-94. If the first item in the operand is omitted, the symbolic name IOCSIXF will be assigned. When an actual index word or a symbolic address is specified, Autocoder will equate the name IOCSIXF to it.

The second item in the operand, IOCSIXG, is used to specify the second IOCS index word for programs using tape files. This item may be a symbolic name or an actual one-digit or two-digit index word address in the range 3-94. If the second item in the operand is omitted, the symbolic name IOCSIXG will be assigned. When an actual index word or a symbolic address is specified, Autocoder will equate IOCSIXG to it.

The third item in the operand, IOCSIXH, is used to specify an IOCS index word for programs using unit record files. This item may be a symbolic name or an actual one-digit or two-digit index word address in the range 3-94. If the third item in the operand is omitted, the symbolic name IOCSIXH will be assigned. When an actual index word or a symbolic address is specified, Autocoder will equate IOCSIXH to it.

The fourth item in the operand, CHANn, is used to specify the number of the highest tape channel to be used by the program. Thus, the programmer would write CHAN1, CHAN2, CHAN3, or CHAN4 to show that the program was to use channel 1, channels 1 and 2, channels 1, 2, and 3, or channels 1, 2, 3, and 4, respectively.

The fifth item in the operand, OPENn, is used to specify the method of handling the OPEN macro-instruction. The value of n may be 1-6. If 1 or 5 is used, the special procedure discussed under "Use of OPEN1" in the "IBM 7070 Input/Output Control System" bulletin should be followed.

1. If OPEN1 is entered in the operand, the OPEN subroutine will not be preserved in storage after it is used; other subroutines will be loaded into the locations

used by the OPEN subroutine. Thus, all tape files must be opened at the same time. The DTFS and File Schedulers must have been loaded into storage before this routine is loaded and executed.

2. If OPEN2 is used, the OPEN subroutine will be retained in storage for use whenever needed.

3. If OPEN3 is entered in the operand, the OPEN subroutine will be written on the tape provided for checkpoint records and read into storage whenever needed. The storage locations required for the OPEN subroutine will be used for other subroutines during the time the OPEN subroutine is on tape. When OPEN3 is specified, EOR1 and CHPT must also be specified.

4. If OPEN4 is used, the OPEN subroutine will be retained in storage for use whenever needed as for OPEN2, except that Form 3 and Form 4 records cannot be processed and three input/output areas cannot be used for one file. The OPEN4 subroutine will occupy fewer storage locations than the OPEN2 subroutine.

5. If the three-area rotating method is used, either OPEN5 or OPEN6 must be specified. OPEN5 and OPEN6 contain provisions for the three-area rotating system; otherwise they are the same as OPEN1 and OPEN2, respectively. The DTFS and File Schedulers must have been loaded into storage before OPEN5 is loaded and executed.

The sixth item in the operand, EORn, is used to specify whether tape labels are to be processed in the End-of-Reel subroutines. The value of n in EORn may be either 1 or 2.

1. The use of EOR1 in the operand specifies that the reading or writing of tape labels is to be determined by the LABELINF entry in the appropriate DTF for each input and output file. EOR1 is required when OPEN3 and/or CHPT is specified.

2. If EOR2 is used in the operand, none of the input tapes may have labels nor will any labels be written on output tapes.

The seventh item in the operand, CHPT, is used to specify whether checkpoint records are to be written. If CHPT is entered in the operand, checkpoint records will be written under the control of the DCHPT statement; EOR1 must also be specified. If CHPT is omitted from the operand, no checkpoint records may be written.

The eighth item in the operand, IGENn, is used to specify the use of SPOOL programs and illegal double-digit character checking in the IOCS tape error subroutine. The value of the final n in IGENn may be 1, 2, 3, or 4.

1. Entering IGEN1 in the operand indicates that a SPOOL program(s) may operate with this main program and that the tape error subroutine is to check for illegal double-digit characters.

2. Entering IGEN2 in the operand indicates that a SPOOL program(s) may operate with this main program but that the tape error subroutine will not check for illegal double-digit characters.

3. Entering IGEN3 in the operand indicates that a SPOOL program will never be run with this main program but that the tape error subroutine is to check for illegal double-digit characters.

4. Entering IGEN4 in the operand indicates that no SPOOL program can be run with this main program nor will the tape error routine check for illegal double-digit characters.

Processing Techniques

The DIOCS statement may appear only once in a program. As noted previously, the first three items in the operand may be used to specify names for certain index words used by the Input/Output Control System. Each of the remaining entries is used by Autocoder to determine the version of the corresponding IOCS subroutine that will be produced. Since the generated material is located at the point where the DIOCS statement is encountered, the programmer should not include the DIOCS statement within a series of imperative statements.

As noted previously, IOCSIXF, IOCSIXG, IOCSIXH, and/or CHPT may be omitted from the operand. In each case, separating commas must be written. Thus, if all four of the above items are omitted, the operand would appear as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DIOCS	,,,CHANn,OPENn,EORn,,IGENn
02			

If a DIOCS operand does not contain eight items (including omitted items indicated by a separating comma) the IOCS subroutines will be generated as though the following DIOCS statement had been entered:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–50)
01	ANYLABEL	DIOCS	,,,CHAN2,OPEN2,EOR1,CHPT,IGEN1
02			

If any of the individual items in a DIOCS statement are invalid, the processor will generate one of the following subroutines corresponding to the invalid item: CHAN2, OPEN2, EOR1, CHPT, or IGEN1. Thus, an incorrect entry as the fourth item in the operand would cause the processor to assume CHAN2, etc.

Error and Warning Messages

CHAN ENTRY INVALID. TWO CHANNELS ASSUMED

If the CHANn entry does not specify CHAN1 or CHAN2, two channel schedulers will be generated.

CHPT ENTRY INVALID. CHPT INCLUDED

If the CHPT entry is neither blank, nor "CHPT," this message will be produced. The checkpoint routine will be generated.

CHPT REQUIRES EOR1. EOR1 GENERATED

If the CHPT entry is not blank and if the EORn entry specifies EOR2, this message will be produced. EOR1 will be generated.

EOR ENTRY INVALID. EOR1 ASSUMED

If the EORn entry does not specify EOR1 or EOR2, EOR1 will be generated.

IGEN ENTRY INVALID. IGEN1 ASSUMED

If the IGENn entry is not in the range IGEN1 to IGEN4, IGEN1 will be generated.

IMPROPER OPERAND. CHAN2, OPEN2, EOR1, CHPT, IGEN1 ASSUMED

If the operand of the DIOCS entry does not contain 8 parameters, this message will be produced. An IOCS package consisting of the items named in the message will be generated.

IOCSIXF(G, H) ENTRY OUT OF RANGE, IGNORED

If any of the index words in the DIOCS entry is specified as actual, and if it does not lie in the range 3-94, this message will be produced. The specified index word will be ignored.

OPEN ENTRY INVALID. OPEN2 ASSUMED

If the OPENn entry is not in the range OPEN1 to OPEN6, OPEN2 will be generated.

OPEN3 REQUIRES CHPT. OPEN2 GENERATED

If the OPENn entry specifies OPEN3 and if the CHPT entry is blank, this message will be produced. OPEN2 will be generated.

DTF—Define Tape File

Each input and output tape file required by a program must be described by a set of File Specifications in a DTF statement. The DTF statement consists of a header line and 35 subsequent entries which describe the file, subroutine locations supplied by the user, and the location of tape label information. These lines are entered on the 7070 File Specifications Coding Sheet shown in Figure 3.

Each DTF statement causes the generation of a nine-word File Specifications Table and a Tape File Scheduler which are used by the Input/Output Control System in conjunction with input and output tape file operations. In addition, the Autocoder compiler used the information furnished by the DTF in order to generate the proper instructions when the name of the file is used in a macro-instruction.

Source Program Format

DTF Header Line

The basic format for the DTF header line is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21 25 30 35 40 45)
01	ANYLABEL	DTF	FILENAME
02			

ANYLABEL may be any symbolic label; if it is omitted, the processor will generate a label. The entry DTF must be written exactly as shown. The operand must contain the name of the tape file defined. This name will be used in the operand of macro-instructions which refer to this tape file.

DTF Subsequent Entries

Although the line numbers for each DTF subsequent entry need *not* be the same as those which appear in the File Specifications Coding Sheet, a card must appear for each line, in the order shown. If a DTF does not consist of exactly 36 entries, the processor will discontinue compilation. A label for each line is not necessary unless the entry will be referred to by the program; the labels of other entries may be blank. The operand for each entry must be as specified in the bulletin, "IBM 7070 Input/Output Control System."

Processing Techniques

Since the Input/Output Control System requires that all File Specifications Tables appear in consecutive locations, the programmer must insure that all DTFS are entered together at the desired point in the program. The processor will determine the location of the first DTF encountered and make this location available to the appropriate input/output routines. A nine-word constant will be assembled at the point where each DTF is encountered.

In addition, the processor will generate a corresponding Tape File Scheduler. This scheduler will be located with literals, adcons and other material generated out-of-line; placement may be determined by use of a LITORIGIN statement.

FIGURE 3

X28-1366

IBM

Program ______________________

Programmed by ______________________

Date ______________________

7070 AUTOCODER CODING SHEET
7070 INPUT/OUTPUT CONTROL SYSTEM
TAPE FILE SPECIFICATIONS

Identification (76–80)

Page No. (1 2) of ______

Line (3–5)	Label (6–15)	Operation (16–20)	Operand (21–75)
01	TAPEFILE	DTF	
02	FCHANNEL		
03	BASETAPE		
04	ALT1TAPE		
05	ALT2TAPE		
06	ACTIVITY		
07	BLOCKCNT		
08	FILEFORM		
09	FILETYPE		
10	RECLNGTH		
11	BLOCKING		
12	OPENPROC		
13	CLSEPROC		
14	TPERROPT		
15	IORDWLST		
16	IOMETHOD		
17	TIOAREAS		
18	PRIORITY		
19	INDXWRDA		
20	INDXWRDB		
21	TDENSITY		
22	SLRPROCD		
23	LLRPROCD		
24	SCLPROCD		
25	TPERRFLD		
26	TPSKPFLD		
27	EOSPROCD		
28	EORPROCD		
29	EOFPROCD		
30	RWDPROCD		
31	CHECKPNT		
32	LABELINF		
33	SRBFORM4		
34	RLIFORM3		
35	SPAREINF		
36	SCHEDINF		

When the processor finds that all consecutive DTFS have been assembled, it will generate a NOP to signal the end of the series of Tape File Specifications Tables. If a separate DTF is encountered at some later point in the program, it will not be included in the series of File Specifications Tables and a warning message will be issued. The DTF will, in general, be assembled properly, but a corresponding File Scheduler will *not* be generated. Therefore, the ninth word of the assembled DTF will not contain the customary address of a File Scheduler. In addition, the assembled DTF will not be available to the Input/Output Control System during the object program unless it is moved to a point within the previously mentioned series of File Specifications Tables and the address of an appropriate File Scheduler inserted in the ninth word.

The product of the blocking factor (BLOCKING, line 11) and the tape input/output areas (TIOAREAS, line 17) in the DTF for a particular tape file determines the number of areas to be specified in the header line of the DA entry which defines the area for the tape file records. (See "N (Area Number)," page 32.)

When the name of an input or output file is to be used in Autocoder macro-instructions which are not a part of the Input/Output Control System, the following requirements must be considered:

1. The name of the file used in the operand of the DTF entry must be identical to the name of the DA entry which defines the areas for the input or output records.
2. The operand of the DA entry which defines the areas must specify an implicit index word and that index word must be the same as the operand of the INDXWRDA entry in the appropriate DTF.

DUF—Descriptive Entry for Unit Records

When unit record files are to be handled by the Input/Output Control System, a DUF entry for each file must be supplied which describes the type of file and the unit record equipment to be used. The DUF entry also supplies the locations of subroutines written by the user that are unique to the file.

Source Program Format

The basic format of the DUF statement is as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50	Basi
01	ANYLABEL	DUF	FILENAME, FILETYPE, CARDSYNC,	
02			LISTADDR, INDXWORD, EOFADDRS,	
03			ERRADDRS	
04				

ANYLABEL is any symbolic label; it may be omitted. The entry DUF must be written exactly as shown.

The first item in the operand, FILENAME, is the name of the unit record file to be described by the DUF entry. This name will be used in the operand of macro-instructions which refer to this unit record file.

The second item in the operand, FILETYPE, is a one-digit number from 1 through 4 to specify the type of unit record file and the operating conditions.

1. A 1 indicates that the unit record file is an input file and that the input unit used by the file will never be shared with a SPOOL program.
2. A 2 indicates that the unit record file is an output file and that the output unit used by the file will never be shared with a SPOOL program.
3. A 3 indicates that the unit record file is an input file and that the input unit may be shared with a SPOOL program.
4. A 4 indicates that the unit record file is an output file and that the output unit may be shared with a SPOOL program.

The third item in the operand, CARDSYNC, is a one-digit number to specify the synchronizer to be used for the unit record file. A 4 must be used when an input file is to be read through an IBM 7501 Console Card Reader.

The fourth item in the operand, LISTADDR, is the address, either actual or symbolic, of the RDW(s) for the unit record area generated by a DA or DRDW statement.

The fifth item in the operand, INDXWORD, is either a two-digit number from 03 through 94 or a symbolic name which specifies the index word to be associated with the unit record file. The indexing portion (positions 2 through 5) of the index word will contain the location of the first word of the current unit record.

The sixth item in the operand, EOFADDRS, is an optional address that may be either actual or symbolic. The address specifies the location of a card reader end-of-file

routine or a printer carriage tape channel 9 routine to be entered if either condition occurs. When this item is omitted and a card reader end-of-file condition occurs, the program will come to a programmed halt if no SPOOL program is to be run in conjunction with it. If a SPOOL program is to operate in conjunction with the program, the program will enter a loop to permit the SPOOL program to continue. When this item is omitted, a channel 9 condition occurring during the printing of a unit record file will have no effect and the program will continue normally.

The seventh item in the operand, ERRADDRS, is an optional address that may be either actual or symbolic. The address specifies the location of an error routine which will be entered when an error occurs during the execution of a macro-instruction which refers to the file named in the first item of the DUF entry. If this item is omitted and an error occurs, the error record will first be typed on the console typewriter. When the file is an output file, the error record will also be punched or printed and processing then resumes automatically. When the file is an input file, the operator may correct the error card immediately and read it in again or he may depress the Start key and resume processing if immediate correction is not required.

Processing Techniques

The DUF entries are entered with the source program when the program is assembled. They should be positioned in the same manner as declarative entries.

As noted earlier, the sixth and seventh items in the operand are optional. Whenever the sixth item is omitted and the seventh item is included, the omission of the sixth item must be indicated by a comma; i.e., two commas will appear between the fifth and seventh items. When both items are omitted, commas *are not* required; only the first five items need appear in the operand.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified.

ACTUAL ADDRESS NOT ALLOWED

This message is issued if FILENAME is not a symbolic name. The entire DUF statement is ignored.

AN ELEMENT OF THE OPERAND STARTS ILLEGALLY

This message is issued in connection with the DUF if either of the following conditions occur:

1. The character following a comma is not alphabetic, numerical, blank or comma.
2. FILENAME begins with a blank.

Processing will occur on all parameters (if any) which appear before the parameter in error; the remainder will be ignored.

LABEL SHOULD BE BLANK

This message is issued if any continuation cards for the DUF operand have an entry in their label column. The entry is processed properly but the label is ignored; if referenced by the program, it will be undefined.

OPERAND OUTSIDE OF ALLOWABLE RANGE

This message is issued if:

1. FILETYPE is not 1, 2, 3, or 4.
2. CARDSYNC is not 1, 2, 3, or 4.
3. INDXWORD, if actual, is greater than 99.

The processor assumes that the value is 1 and normal processing continues.

STMNT SHOULD OR SEEMS TO BE ENDED BUT CARDS REMAIN

This message is issued if there are continuation cards remaining but the current card includes the seventh parameter or ends with a blank. The remaining cards are ignored.

SYMBOLIC ADDRESS NOT ALLOWED

Issued if FILETYPE or CARDSYNC begins with an alphabetic character. Preceding parameter(s) will be processed; the remaining parameters are ignored.

DA—Define Area

The declarative statement, DA, may be used to define and reserve any portion of storage. An area may be reserved for use as an input, output, or work area, or contiguous areas may be reserved to contain a number of records, all of which are identical in format. The DA statement instructs the processor concerning the positions, lengths, and names of fields which make up the record area(s) being defined, as well as the characteristics of the data which is to occupy each field. Such characteristics include format, implied decimal-point position, and double- or single-digit representation. Locations, field definers, and, if specified, implicit indexing are assigned by the processor to enable the programmer to refer to the fields by name in imperative statement operands. Thus, the programmer need not be concerned with the actual locations of the fields within storage. It should be remembered that the DA statement does not provide the data for the field it defines, but only reserves the space which the data is to occupy. The data itself must be brought into the defined area in core storage from some external source such as cards or magnetic tape, or from other locations in storage.

A DA statement consists of a header line and one or more subsequent entries. The DA header line is used to *initiate* the reservation of a portion of storage. The header line specifies the number of identical record areas to be reserved. An indication to generate RDWs corresponding to the areas may be specified. Relative addressing and implicit indexing, which facilitate referencing a field within a record area, may also be specified in the DA header line. The subsequent entries define the fields within the record area, specify the amount of storage to be reserved, and describe the data which will appear in each field.

Source Program Format

DA Header Line

The basic format of the DA header line is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–50)	Basic Au
01	ANYLABEL	DA	N,±RDW,ADDRESS±ADDRADJ+INDEXWORD	
02				

ANYLABEL is any symbolic label; it may be omitted. The operation code, DA, must be written exactly as shown. With the exception of the first entry, N, the items in the operand are optional. An explanation of the entries in the operand follows.

N (Area Number). N, the first entry in the operand of the DA header line, may *not* be omitted. N is replaced by the number of identical record areas to be reserved by the DA operation, the format of which is defined by the programmer. For example, suppose that records of identical format are to be read into storage in blocks of ten. The programmer might enter a DA header line with N equal to 10, followed by subsequent entries which specify the starting and ending posi-

tions, characteristics, and names of the fields composing one record. If all other items in the operand were omitted, the DA header line would appear as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DA	10
02			

N may be any unsigned number from 1 to 9999; it is limited only by the size of storage. (Frequently, N will equal 1, as in a work area or a unit record input or output area.) The number of storage words to be reserved for the entire DA area (excluding RDWs, if any) will be N times the number of words reserved for the one record defined by the subsequent entries. The maximum number of words which may be reserved for an entire DA area is also limited only by the size of storage.

If the DA is to reserve contiguous areas of storage that are to contain a number of input or output records having identical format, then N must equal the product of the blocking factor and the tape input/output areas indicated in the DTF for the file (see page 28). If the blocking factor for a tape file is 10 and the programmer only wishes to use one area of storage for input or output, then N should equal 10. However, if the programmer wishes to use two consecutive areas of storage for input or output, then N must equal 20 and if he wishes to use three consecutive areas, then N must equal 30.

RDW (Record Definition Words). Record definition words are required by the 7070 and the 7074 for reading in and writing out data and for moving blocks of data within storage. If "RDW" is written in the operand of the DA header line, the processor will automatically generate N RDWs associated with the N defined areas. These RDWs will be assigned N locations immediately preceding the first word of the first area defined.

If RDW is not preceded by a sign, all generated RDWs will be plus except the last, which will be minus. If a + or − sign precedes RDW, *all* generated RDWs will be given the indicated sign.

If RDW is not written in the operand, RDWs will not be generated; the first word reserved by the DA statement will be the first word of the first record area.

For example, suppose that an area is to contain a group of four 10-word records and that the following DA header line is entered, the fields within a record being defined by subsequent entries:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	RECORD	DA	4,RDW
02			

Assume that Autocoder's location assignment counter contains 1000 when the above DA statement is encountered. The processor will generate the following

RDWs associated with the four record areas and assign them to locations immediately preceding the first record area:

Symbol	*Location*	*RDW*
RECORD	1000	+0010041013
	1001	+0010141023
	1002	+0010241033
	1003	−0010341043

Note that the first RDW, located at 1000, defines the first 10-word record area beginning at 1004; the second RDW, located at 1001, defines the second 10-word record area beginning at 1014; etc. If RDW had been omitted from the DA header line, the first word of the first 10-word record would have been located at 1000; no RDWs would have been generated.

RDWs may also be defined elsewhere by using the DRDW statement.

ADDRESS (Relative Addressing). The fields defined in subsequent entries may be assigned addresses relative to (i.e., beginning with) an ADDRESS specified in the operand of the DA header line. ADDRESS may be written in actual, *, or symbolic form. When a relative address is specified, it will usually be 0.

Assume that a relative address of 0 is specified, as in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DA	20,RDW,0
02			

In this case, any field which occupies the first, second, or third word, etc., of the area beyond the generated RDWs will be assigned relative addresses of 0000, 0001, 0002, etc., respectively. Thus, if FIELDC is the name of a field occupying the third word of each record, the instruction

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		ZA1	FIELDC(0,4)+X11
02			

would be translated in the object program as +131104*0002*. Since twenty identical areas have been reserved and since the third word of each of the twenty areas is addressed FIELDC and assigned the relative address 0002, one of the twenty areas referred to by an instruction must be specified. The desired area may be indicated by indexing the individual instruction (as illustrated) or by implicit indexing (as explained in the next section). Referring to the illustration, the indexing portion of index word 11 must have been programmed to contain the starting location of the area to be processed. Then, when indexing takes place, the indexing portion of index word 11 is added algebraically to the relative address of FIELDC, 0002, resulting in the actual address of FIELDC.

The presence of a relative address in the DA header line does not affect the reservation of storage. The generated RDWs define the actual record areas reserved; they are also unaffected by the absence or presence of relative addressing. For example, assume that the location assignment counter is at 1000 when the following DA header line is encountered:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DA	20,RDW,0
02			

Twenty RDWs associated with the twenty record areas will be assigned locations 1000 through 1019. If the subsequent entries define a record that is ten words in length, the first RDW generated at 1000 would be +0010201029 and the first word of the first record would be assigned to 1020, the second to 1021, etc.

If relative addressing is specified, but RDWs are not to be generated, commas must appear in two consecutive columns, as in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DA	20,,ADDRESS+INDEXWORD
02			

ADDRADJ (Address Adjustment). Address adjustment is permitted with a relative address in symbolic or * form. However, when a relative address is specified, it will usually be 0 and address adjustment will not appear.

Address adjustment, when used with an * or symbolic address, will appear as in the following examples:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	AREANAME	DA	3,RDW,ADDRESS-10+INDEXWORD
02		.	
03		.	
04		.	
05	RECORD	DA	100,,*+1+INDEXWORD
06			

INDEXWORD (Implicit Indexing). An index word, either symbolic or actual, may be specified in a DA header line as in the following examples:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	AREANAME	DA	15,,RDW,,ADDRESS+INDEXWORD
02		.	
03		.	
04		.	
05	INPUTFILE	DA	13,,,ADDRESS−1+INDEXWORD
06		.	
07		.	
08		.	
09	OUTPUTFILE	DA	20,,,ADDRESS+X15
10		.	
11		.	
12		.	
13	CHANGEFILE	DA	8,,RDW,,ADDRESS+10+X7
14			

In these examples, INDEXWORD is the symbolic name of an index word; X15 and X7 are the actual addresses of index words. Note that a plus sign must always precede the name of the index word and that actual numbers of index words must be preceded by "+X." The specification of indexing follows address adjustment, if present, or the relative address itself, if address adjustment is not used.

The naming of an index word in the header line has no effect upon the storage reservation and location assignments produced by the DA. It does facilitate writing instructions which reference fields in one of the records within the defined area. When the compiler encounters an instruction referencing such fields, the instruction will be examined for the presence of indexing. If indexing already appears in the instruction, it will be left unchanged. If indexing does not appear in the instruction, however, the address of the index word specified in the DA header line will be inserted in the index word positions of the instruction. Thus, if it is known when coding a DA that the record defined by it is to be processed by indexing, the indexing may be caused by making the single notation in the DA header line, rather than by supplying the indexing in every instruction which acts upon the record.

Consider an example in which the compiler encounters the following DA header line:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	AREANAME	DA	100,,,0+X25
02			

Each field in a record will be assigned a location relative to 0. If FIELD2 is the label of a field occupying the second word of each record in the area, the instruction

Line	Label	Operation	OPERAND
01		ZA1	FIELD2
02			

will result in an assembled machine instruction of +1325090001. Notice that the 6-9 portion of the instruction is 0001, but that this address is to be indexed by index word 25. The indexing portion of index word 25 must have been programmed to contain the starting location of the area to be processed. Then, by indexing the instruction by index word 25, the indexing portion of the index word is added algebraically to the relative address of FIELD2, 0001, resulting in the actual address of FIELD2.

If, however, the field had been referred to by an instruction which already included indexing, such as

Line	Label	Operation	OPERAND
01		ZA1	FIELD2+X29
02			

the assembled instruction would have been +1329090001, thus ignoring the specification of index word 25 in the DA header line.

As demonstrated by the previous examples, the use of a DA header line to specify a relative address and implicit indexing is most valuable in processing blocked tape records. If the fields of a record are defined as relative to 0 and the RDWS associated with the records are successively loaded into the index word specified in the DA header line, instructions to act upon the fields of the record may be written as if no indexing was required. Autocoder will insert the address of the index word into the index word positions of all such instructions.

For another example, assume that an area is to contain a group of four 10-word records. Assume also that it is desired to perform the following operation on three fields named FIELDA, FIELDB and FIELDC, which occupy the first, second and third words, respectively, of each record: subtract the contents of FIELDB from the contents of FIELDA and store the result in FIELDC. This could be accomplished by the following coding:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	INAREA	DA	4,+RDW,0+X2
02		•	
03	(Subsequent entries define		
04	the fields within a record.)		
05		•	
06		XL	3,+0000000003
07	LDRDW	XL	2,INAREA+X3
08		ZA1	FIELDA
09		S1	FIELDB
10		ST1	FIELDC
11		BIX	3,LDRDW
12			

Assume that Autocoder's assignment counter contains 1000 when the DA statement is encountered, 2000 when the first instruction of the routine is encountered, and the literal +0000000003 has been assigned location 4432. The corresponding assembled machine-language instructions would be as shown below:

Location	*Assembled Instruction*
2000	+4500034432
2001	+4503021000
2002	+1302090000
2003	−1402090001
2004	+1202090002
2005	+4900032001

DA Subsequent Entries

Subsequent entries under a DA header line are used to name fields, to indicate their starting and ending positions within the record area, and to specify their format. The subsequent entries, taken collectively, indicate a logical assemblage of words, the number of which is multiplied by the number N in the DA header line to determine the amount of storage (excluding RDWs, if any) to be reserved by the DA.

The label column of a DA subsequent entry may include any symbolic name or may be left blank (except as noted for the special operation, CODE, which is explained on page 50). The operation column is left blank, except when used for the CODE entry.

Field Position and Length. The position and length of a field within the record area is indicated in the operand by writing the starting digit position, a comma, and the ending digit position. If the area is considered to be a consecutive string of digits, the first being 0, the position of any digit is identical to its placement in the string. Thus, a field occupying the entire first word of a record area would be defined as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	AREANAME	DA	1
02	FIELDA		0,9
03			

In the above example, 0 is the digit position of the first character and 9 is the digit position of the tenth and final character in FIELDA.

A field occupying the entire second word of a record area would be defined as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	AREANAME	DA	1
02	FIELDB		10,19
03			

A field occupying the last two digits of the third word of a record area would be defined as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	AREANAME	DA	1
02	FIELDC		28,29
03			

A digit position, therefore, is a combination of a word number (consisting of one, two, three, or four numerical characters) followed immediately by a digit number (always one numerical character). The first word of the area is referred to as word 0, the second as word 1, the third as word 2, etc.; digits within a word are numbered 0 through 9, left to right, in the normal fashion. Thus, digit position 49 would refer to the low-order digit in the fifth word of a record area; digit position 9990 would refer to the high-order digit in the thousandth word of a record area.

It is not necessary to enter leading zeros; the following operands would have exactly the same effect:

12,25
00012,00025

A one-digit field may be entered by writing the single digit position only. Thus, the following operands would have exactly the same effect:

19,19
19

In either case, the processor would reserve the low-order digit position of the second word of the record area.

The following example provides additional illustrations of entries which define the length and position of fields within an area.

Line	Label	Operation	OPERAND
01	RECORD	DA	1,RDW
02	NAMEONE		00,16
03	NAMETWO		17,19
04	NAMETHREE		20,29
05	NAMEFOUR		53,57
06	NAMEFIVE		58
07			79

An eight-word record is defined (digits 0-79). The first field, NAMEONE, occupies a total of 17 digit positions beginning with the high-order position of the first word of the record. NAMETWO and NAMETHREE are fields of lengths 3 and 10 digits, respectively.

A gap of 23 digits then occurs, followed by NAMEFOUR, a five-digit field. This is followed by NAMEFIVE, a one-digit field. (Since the starting and ending digit positions are identical, the one number, 58, suffices.)

A subsequent entry is always required when a field is to be named for reference by the object program. Other fields in a record need not be defined by a subsequent entry except for an entry to terminate the record. The field *must* appear but need not have a label. Thus, in the example above, the one-digit field in digit position 79 is not named, but it is necessary in order to establish that eight words are occupied by the record. Any digit in the eighth word (73, for example) would cause the DA area to be comprised of eight full words, since a record is never terminated in the middle of a word.

Subsequent entries may be entered in any order. However, fields are normally entered in ascending storage-position order for convenience and accuracy.

The net effect of the following coding would be identical to that of the preceding example, despite the order of the subsequent entries:

Line	Label	Operation	OPERAND
01	RECORD	DA	1,RDW
02			79
03	NAMEONE		00,16
04	NAMEFIVE		58
05	NAMETWO		17,19
06	NAMEFOUR		53,57
07	NAMETHREE		20,29
08			

Subfields are entered in the same DA which defines the fields. Any desired breakdown of the fields may be indicated by writing the starting and ending digit positions of subfields which overlap, fall within, or bridge other fields. Again, the order is irrelevant. The following example illustrates the coding of subfields:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	CALENDAR	DA	1,RDW
02	DATE		74,79
03	DAY		76,77
04	YEAR		78,79
05	MONTH		74,75
06	DECADE		78
07			

In this example, DATE will reference the entire date field; the subsections of the date field may be referred to as MONTH, DAY, or YEAR. In addition, DECADE references a one-digit field within YEAR and also, therefore, within DATE.

Relative Field Definition. The primary function of the DA declarative operation is to instruct the processor as to the positions and lengths of fields within an area, thereby allowing the processor to assign storage locations and field definers automatically when a field is referred to by name in the operand of an instruction. Sometimes it may be desirable to refer to a portion of a field which has been defined by a declarative operation. For the convenience of the programmer, this is done relative to the field itself, so that it is not necessary to remember the actual digit positions of the field.

To illustrate, assume that the name CUSTNO is a field of seven digits, defined under a DA as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	INPUTAREA	DA	1
02	CUSTNO		03,09
03			

A field might occupy only the low-order digit positions of a word, as does CUSTNO above. However, positions within the field itself are numbered starting with 0. If the programmer wishes to refer to the three high-order positions of CUSTNO, he would refer to positions 0, 1 and 2 by placing the entry CUSTNO (0, 2) in the operand. During assembly, the processor will convert the field definers relative to CUSTNO to field definers which refer to actual digit positions of a word. Thus, during assembly, the processor will convert CUSTNO (0, 2) to the actual digit positions 3, 4 and 5 of the word of which CUSTNO is a portion. If the storage location of this word is 1001, symbolic instructions using CUSTNO and their assembled equivalents might be as follows:

Symbolic Instruction	*Assembled Instruction*
ZA1 CUSTNO(0, 2)	+1300351001
ZA1 CUSTNO(4, 5)	+1300781001
ZA1 CUSTNO	+1300391001

CUSTNO might instead be defined as a full word, as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	INPUTAREA	DA	1
02	CUSTNO		00,09
03			

In this case, any field definition following CUSTNO would be both relative and actual. If the storage location of this word is 1001, symbolic instructions using CUSTNO and their assembled equivalents might be as follows:

Symbolic Instruction	*Assembled Instruction*
ZA1 CUSTNO(0, 2)	+1300021001
ZA1 CUSTNO(4, 5)	+1300451001
ZA1 CUSTNO	+1300091001

In the example on page 41 illustrating how subfields are entered, the subfield DAY would be equivalent to DATE(2, 3).

Format Indicators. In addition to defining the position and length of a field, a DA subsequent entry may also specify format and characteristics for that field. It will be noted that Autocoder macro-instructions are capable of acting upon fields which bridge words, which exceed ten digits in length, or whose decimal points are not aligned. The macro generators will consider the formats for the fields concerned; they will then generate instructions to align decimal points while performing arithmetic and compare operations, and will convert a field from one format to another format.

Five different types of fields may be designated by writing the proper format indicators immediately after the ending digit position of the field. The following indicators may be used to indicate to the processor the format of the data that will occupy the field at object program time.

Format Indicator	*Meaning*
A	The field is to contain numerical data to be treated as an automatic-decimal number. Although automatic-decimal fields must not exceed 20 digits for arithmetic operations, longer integer fields may be defined, and are acceptable to some macro-instructions.
F	The field is to contain numerical data to be treated as a floating-decimal number. The field must be exactly 10 digits in length, beginning in digit position 0 of a word; it may *not* bridge words.

Format Indicator	Meaning
@	The field is to contain the double-digit representation of alphameric characters. The data is to be treated as neither an automatic-decimal nor a floating-decimal number. The field may be any desired length; the number of digits in the field must be evenly divisible by two because each character is represented by two digits. The field must start at an even-numbered digit position.
@A	The field is to contain the double-digit representation of data which is to be treated as an automatic-decimal number. The size of the field must not exceed 40 digits (20 characters). The number of digits in the field must be evenly divisible by two because each character is represented by two digits. The field must start at an even numbered digit position.
@F	The field is to contain the double-digit representation of data which is to be treated as a floating-decimal number. The size of the field must be exactly 20 digits, occupying two full words; it may not be split among three words.

If format indicators are omitted, the processor will assume that the field is to contain numerical data to be treated as a signed integer:

1. For fields less than or equal to 20 digits in length, this is equivalent to an automatic-decimal field.
2. For fields longer than 20 digits, see individual macro-instructions for the treatment of long integer fields.

Users are strongly urged, however, to furnish format indicators in DA subsequent entries.

Decimal-Point Indicators. In the case of automatic-decimal numbers, indicators may also be used to indicate where the decimal point would fall if it were included in the field. (It should be noted that the decimal point is thus implicit, or "understood"; it is not actually to be included in the field nor is a field position to be reserved for it.) The implicit positioning of the decimal point is done by placing an indicator of the form

a.b

immediately to the right of the A or@A format indicator. The "a" is replaced by the number of characters to the left of the implicit decimal point in the field described; the "b" is replaced by the number of characters to the right of the implicit decimal point. The sum of these indicators must not exceed 20 characters in length and must exactly equal the number of characters defined by the starting and ending positions for the field. For a numerical field, then, $(a + b)$ must equal the number of digits defined for the field; for a double-digit representation, however, $(a + b)$ must equal one half of the number of digits defined for the field.

For example, the operand

20,25A4.2

informs the processor that the field which occupies digit positions 20 through 25 should be operated upon as if it included six digits of numerical data and a decimal point fell between the four high-order digits and the two low-order digits.

However, the operand

20,25@A2.1

informs the processor that the field should be operated upon as if a decimal point fell between the two high-order *characters* and the low-order *character* of a *three-character* alphameric field. (It will be remembered that one character occupies two digit positions.)

When an integer is defined, the decimal point and the number of decimal positions may be omitted. Thus, the following entries would have the same effect:

20,29A10.0
20,29A10

If the A or @A format indicator is not followed by decimal-point indicators, the processor will assume that the field is to contain an integer. Users are strongly urged, however, to furnish format indicators and decimal-point indicators for automatic-decimal numbers in DA subsequent entries. Specification of these characteristics makes it possible for macro generators to perform diagnostic analysis of the source-program statements, and thus aid in minimizing problems in program testing.

Additional examples of the use of format and decimal indicators are given below:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	INPUT	DA	1,RDW
02	FIELDA		10,15A4.2
03	FIELDB		24,29@
04	FIELDC		30,39F
05	FIELDD		46,55@A3.2
06	FIELDE		60,79@F
07	FIELDF		80,85A6.0
08	FIELDG		86,89@A2.0
09			

FIELDA reserves the six high-order positions of the second word of the record area for an automatic-decimal number which will have, at most, four integer places and two decimal places.

FIELDB reserves the six low-order positions of the third word of the record area for a maximum of three alphameric characters.

FIELDC reserves the entire fourth word of the record area for a floating-decimal number.

FIELDD reserves the last part of the fifth and the first part of the sixth words of the record area for an automatic-decimal number in double-digit form, having

at most three integer places and two decimal places. Ten digit positions must be reserved, however, because of the double-digit representation.

FIELDE reserves the seventh and eighth words of the record area for a floating-decimal number in double-digit representation.

FIELDF reserves the six high-order positions of the ninth word of the record area for an automatic-decimal integer.

FIELDG reserves the four low-order positions of the ninth word of the record area for the double-digit representation of an automatic-decimal integer.

Two or more symbolic names may be assigned to the same field and the same characteristics by listing both names as subsequent entries under the same DA and duplicating the operand desired, as in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DA	10,RDW,0+INDEXWORD
02	FIELDA		0,9A6,4
03	A		0,9A6,4
04	FIELDB		20,29F
05	B		20,29F
06	FIELDC		42,59@
07	C		42,59@
08			

In this example, A is made equivalent to FIELDA, B is made equivalent to FIELDB, and C is made equivalent to FIELDC. Any reference to A, B, or C will give precisely the same results as those which would result from reference to FIELDA, FIELDB, or FIELDC, respectively.

The contents of a given field may be treated as having different characteristics at various times by defining that field using more than one subsequent entry, as in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DA	10,RDW,0+INDEXWORD
02	FIELDA		0,9A6,4
03	A		0,9A8,2
04			

Each reference to the field must be to the label which is associated with the characteristics that apply to the contents of the field at that particular time in the object program.

Processing Techniques

Use of DA With Symbolic Machine Instructions

Although the programmer will usually find it more efficient to use macro-instructions to refer to DA header lines and subsequent entries, symbolic machine instructions may also be used for this purpose. When a symbolic machine instruc-

tion references a symbolic label, however, it will operate directly on a maximum of a single full word. Certain additional restrictions must also be considered; these are noted below.

DA Header Line. When a symbolic machine instruction references the label of a DA header line, it will operate on the entire first word reserved by the DA statement. Thus, if RDW generation is specified, the symbolic machine instruction will act on the entire word containing the first generated RDW. If RDW generation is *not* specified, the symbolic machine instruction will act on the entire first word of the first record defined. Symbolic machine instruction references to the label of a DA header line are not affected by the presence or absence of implicit indexing.

The above usages are illustrated by the examples on page 47.

DA Subsequent Entry. When a symbolic machine instruction references the label of a DA subsequent entry, it will act only on that portion of the field which falls within the word containing the starting digit position defined by the subsequent entry. The specification of implicit indexing will cause the symbolic machine instruction to address a word (or portion of a word) in the current record, as determined by the contents of the specified index word. If implicit indexing is not specified, the word (or portion of a word) addressed will be within the first record as defined by the subsequent entry.

The above usages are illustrated by the examples on page 47.

Use of DA with Macro-instructions

The Autocoder language is designed to allow record processing to be accomplished without regard to the number of records that may appear in one block of input or output. Thus, when a macro-instruction refers to the label of a DA header line, *one complete record* will be the maximum that is affected or considered. (It should be noted that, in some cases, only part of the record will be affected or considered.) When N is greater than 1, therefore, no single reference by a macro-instruction to the label of a DA header line will ever affect or consider the *entire* area reserved.

Regardless of the absence or presence of specified RDW generation in the DA header line, the effect of a macro-instruction reference to the label of a DA header line or subsequent entry will be as follows:

1. When a DA header line does *not* contain a relative address and implicit indexing,

   (a) a macro-instruction reference to the label of the DA header line will cause the generation of coding which will affect or consider (as a maximum) the *first* record, as defined by the subsequent entries under the DA header line.

   (b) a macro-instruction reference to the label of a subsequent entry will cause the generation of coding which will affect or consider (as a maximum) the named field within the *first* record, as defined by the subsequent entries under the DA header line.

   This technique would normally be used when a work area is defined; i.e., N is 1 and the first record is, therefore, the only record defined.

```
PAGE AA    PROGRAM DA                                       7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                            CDNO FD  LOC  INSTRUCTION     REF

01 0002   *   EFFECTS OF SYMBOLIC MACHINE INSTRUCTION REFERENCE TO
02 0003   *   LABEL OF DA HEADER LINE.
03 0004   *
04 0005   AREA1      DA    1                                                                +0003250327
05 0006                    12,25                                                    29  0326                 0326
06 0007              ZA1   AREA1                                              00001     0328 +1300090325
07 0008   *
08 0009   AREA2      DA    2                                                                +0003290334
09 0010                    12,25                                                    29  0330                 0330
10 0011              ZA1   AREA2                                              00002     0335 +1300090329
11 0012   *
12 0013   AREA3      DA    2,RDW                                                            +0003360343
13       X                                                                              0336 +0003380340     0336
14       X                                                                              0337 -0003410343     0337
15 0014                    12,25                                                    29  0339                 0339
16 0015              ZA1   AREA3                                              00003     0344 +1300090336
17 0016   *
18 0017   AREA4.     DA    2,RDW,0+INDEXWORD                                                +0003450352
19       X                                                                              0345 +0003470349     0345
20       X                                                                              0346 -0003500352     0346
21 0018                    12,25                                                    29  0348                 0001
22 0019              ZA1   AREA4                                              00004     0353 +1300090345
23 0020   *
24 0021   AREA5      DA    2,,0+X15                                                         +0003540359
25 0022                    12,25                                                    29  0355                 0001
26 0023              ZA1   AREA5                                              00005     0360 +1300090354
27 0024   *
```

```
PAGE AA    PROGRAM DA                                       7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                            CDNO FD  LOC  INSTRUCTION     REF

01 0028   *   EFFECTS OF SYMBOLIC MACHINE INSTRUCTION REFERENCE TO
02 0029   *   LABEL OF DA SUBSEQUENT ENTRY.
03 0030   *
04 0031              DA    1                                                                +0003250327
05 0032   FIELDA           12,25                                                    29  0326                 0326
06 0033              ZA1   FIELDA                                             00001     0328 +1300290326
07 0034   *
08 0035              DA    2                                                                +0003290334
09 0036   FIELDB           12,25                                                    29  0330                 0330
10 0037              ZA1   FIELDB                                             00002     0335 +1300290330
11 0038   *
12 0039              DA    2,RDW                                                            +0003360343
13       X                                                                              0336 +0003380340     0336
14       X                                                                              0337 -0003410343     0337
15 0040   FIELDC           12,25                                                    29  0339                 0339
16 0041              ZA1   FIELDC                                             00003     0344 +1300290339
17 0042   *
18 0043              DA    2,RDW,0+INDEXWORD                                                +0003450352
19       X                                                                              0345 +0003470349     0345
20       X                                                                              0346 -0003500352     0346
21 0044   FIELDD           12,25                                                    29  0348                 0001
22 0045              ZA1   FIELDD                                             00004     0353 +1301290001
23 0046   *
24 0047              DA    2,,0+X15                                                         +0003540359
25 0048   FIELDE           12,25                                              -     29  0355                 0001
26 0049              ZA1   FIELDE                                             00005     0360 +1315290001
27 0050   *
```

2. When a DA header line contains a relative address and implicit indexing,

   (a) a macro-instruction reference to the label of the DA header line will cause the generation of coding which will affect or consider (as a maximum) the *current* record (as determined by the contents of the implicit index word at the time the instructions are executed).

   (b) a macro-instruction reference to the label of a subsequent entry will cause the generation of a coding which will affect or consider (as a maximum) the named field within the *current* record (as determined by the contents of the implicit index word at the time the instructions are executed).

   The generated instructions will include implicit indexing and references to the implicit index word wherever necessary.

Input/output macro-instructions may indirectly affect a record by operating on the RDWS defining that record; e.g., the PUTX macro affects records by interchanging RDWS. When the other macro-instructions (i.e., those not concerned with input/output) refer to the label of a DA header line which specifies RDW generation, however, they will not operate on the RDWS themselves. For example, when the ZERO macro is used in this way, it will not zero the RDWS; it will zero out one complete *record* area.

When input or output files are to be referred to by input/output macro-instructions, the following considerations apply:

1. Macro-instructions which are to operate on a *record* will reference the name of the file. This name must appear as the label of the DA header line *and* as the operand of the DTF for that particular file. Depending on the method of generating RDWS, this name may or may not be the same as the operand of the subsequent entry IORDWLIST in that same DTF.
2. In the header line of the DA referenced, N and RDW specification will be prepared according to the procedures described in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," form J28-6033-1.
3. A relative address of 0 will be specified in the DA header line.
4. Implicit indexing will be specified in the DA header line. The name of the index word specified must be the same as the operand of the subsequent entry INDXWRDA in the appropriate DTF.

When input or output files are to be referred to by the other Autocoder macro-instructions or by *both* the input/output and other macro-instructions, the following considerations apply:

1. If RDWS are generated by the DA header line, all macro-instructions can reference the name of the file. If RDWS are generated by a DRDW, the name of the file cannot be referenced; however, the area which the DRDW references may be used as an operand of any macro-instruction, where applicable.
2. In the header line of the DA referenced, N and RDW specification will be prepared according to the procedures described in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," form J28-6033-1.
3. A relative address of 0 will be specified in the DA header line.
4. Implicit indexing will be specified in the DA header line. The name of the index word specified must be the same as the operand of the subsequent entry INDXWRDA in the appropriate DTF.

When a DA statement is used to define a work area, the following considerations will normally apply:

1. N will be 1.
2. RDW generation will be specified.
3. Neither a relative address nor implicit indexing will be specified.

Arrangement of Fields

In general, the arrangement of fields defined by DA subsequent entries is determined by prior considerations, e.g., the record form and format. Thus, a tape record is usually arranged to take full advantage of the zero suppression of the IBM 7070/7074; i.e., numerical fields which most frequently contain leading zeros are each placed in the high-order positions of a word, but numerical fields which less frequently (or never) contain leading zeros are each placed in the low-order positions of a word. In the interests of tape capacity and speed, it is also desirable to hold the appearance(s) of the delta character to a minimum. In general, this means that all alphameric fields should appear together.

Since imperative statements frequently alter the sign of a word to reflect the sign of the last field entered, the programmer should insure that fields which occupy the same storage word are to be associated with the same sign. If this is not the case, inconsistent results may occur. For example, if alpha information is entered into a word in which another field contains numerical data, the sign of the word may become alpha; the numerical information would then become alpha, possibly consisting of invalid double-digit codes.

When the format indicator F is used, the field must occupy one complete word; when the format indicator @F is used, the field must occupy two complete words.

Within the considerations listed above, the use of macro-instructions allows the programmer extreme flexibility in arranging the fields within a record area. In fact, the use of macro-instructions permits the programmer to refer, with equal ease, to fields which overlap or bridge words and to fields which occupy all or part of a word. It should be noted, however, that the most efficient coding will result when the amount of word-bridging is held to a minimum.

Additional Examples

Additional examples of the use of the DA statement in conjunction with other statements are included throughout the manual.

CODE

CODE is a special type of declarative statement which may *only* appear as one of the subsequent entries under a DA header line. It has the following uses:

1. To name a field in which one or more code values may be present during the running of the object program.
2. To name and define these code values which are the condensed representations of various conditions, categories or classifications, etc.

The primary use of CODE is in connection with the LOGIC and DECOD macro-instructions. By means of these macro-instructions, a code field may be interrogated for the presence or absence of a specific code value. Various switches may be set or branches may be made depending upon the decision. When CODE is used, interrogation may be made by referencing the *symbolic name* of the code value for the condition to be tested, rather than the code value itself. CODE symbolic names may not be referred to in symbolic machine instructions.

Source Program Format

The CODE entry is itself a header line, giving the position and format of the code field. Subsequent entries consist of the symbolic names of the conditions to be tested during the course of the program, each indented one space, with the actual code value indicated for each condition.

A CODE entry may be followed by subsequent fields of the DA or by another CODE entry. If CODE is the last DA entry, any other type of entry may follow.

Code fields may not be more than ten numerical digits or five alphameric characters in length and must be wholly contained within one word of the record defined by the DA.

CODE Header Line

The basic format of the CODE header line is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21 … 25 … 30 … 35 … 40 … 45)
01	ANYLABEL	CODE	20,,29A8.2
02			

A symbolic label must always be supplied for the CODE entry and the letters CODE must be written in the operation column. The operand includes field definition as in any other DA subsequent entry, except that the area defined may not be more than a single storage word and it may not bridge words.

Two types of format indicators (shown below) are allowed. The desired indicator is written immediately after the terminal digit position of the field definition.

Code Format Indicators	*Meaning*
A	The field is to contain numerical data which is to be treated as an automatic-decimal number.

Code Format Indicators	Meaning
	The format indicator may be followed by a decimal position indicator of the form "a.b" as described on page 43. In this case, however, the field size may not be greater than 10.
@	The field is to contain the double-digit representation of alphameric characters. The field size defined must not exceed 10 digits (5 characters); it must be evenly divisible by 2. The starting digit position must be an even number.

If a format indicator does not appear immediately after the ending digit position in the CODE header line, the processor will consider the field to contain an automatic-decimal integer.

Consider the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DA	1
02	ANYLABEL	CODE	20,,29A8.2
03			

The CODE header line defines a field occupying the entire third word of the record area. At object program time, the field is to contain numerical data to be treated as an automatic-decimal number with 8 integer digits and 2 decimal digits.

The following example defines a field which is to contain one alphameric character.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DA	2,,,0+INDEXWORD
02	ANYLABEL	CODE	14,,15@
03			

CODE Subsequent Entries

A symbolic label is always required for a CODE subsequent entry so that the code value may be referenced by name. For this entry, Autocoder requires that the *label* be indented exactly one space. Thus, the maximum size of this type of label is nine characters. The operation column must be blank.

The operand is used to define a code value which may appear in the field during the running of the object program. A code value may not exceed the length of the field as defined by the CODE header line; the maximum length, therefore, is 10 numerical digits or 5 alphameric characters. Each code value must also be consistent with the format indicator(s) which appear in the CODE header line.

If the CODE header line specifies an automatic-decimal field, therefore, the operand of each CODE subsequent entry under that header line must contain a numerical

value. This value may be signed or unsigned. (The sign, however, is superfluous; the digits will be considered absolute in sign. The sign of the word will not be tested or changed.)

In the example below, the CODE header line defines a field which is to contain data to be treated as a one-digit automatic-decimal integer. Each subsequent entry, therefore, must define a one-digit numerical code value.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DA	1
02	COLOR	CODE	25A1
03	BLACK		0
04	GREEN		1
05	RED		5
06	WHITE		6
07	BLUE		8
08			

In this example, COLOR is the name of the defined field. In the source-language program, the field may be tested for the presence of a color such as BLACK. The processor, however, will generate coding which will cause the object program to test for the presence of a "0." If the field is to be tested for the presence of GREEN, the processor generates coding to test for a"1." Thus, the programmer may make reference by means of a macro-instruction to the condition or name of a code value and the processor will translate this into instructions which actually check for the presence of the code value itself.

In the example below, CLASS is the name of a field which is to contain a three-digit, automatic-decimal number. In this case, the number is to include two integer digits and one decimal digit. Each subsequent entry, therefore, must define a numerical code value containing the number of decimal digits specified by the CODE header line. This is done by inserting a decimal point at the proper position within the code value. The decimal point does not occupy a storage location; it merely indicates the treatment of the integer and decimal digits in the code value.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	AREANAME	DA	1
02	CLASS	CODE	101,103A2.1
03	A		11.9
04	B		10.9
05	C		9.9
06	D		8.9
07			

Although the code value must contain exactly the same number of decimal digits as defined in the CODE header line (in the above example, one decimal digit),

leading zeros may be omitted from the integer digits, as in the code values for C and D above.

When CODE defines an alphameric field, the actual code values pertinent to the various conditions should be indicated in the operand by writing the @ character, followed by the alphameric characters which comprise the code value, followed by another @ character. The code value may *not* include the @ character itself; the form @@@ or @A@B@C@ is thus prohibited. In addition, the record mark may *not* be used as a code value. Except for being restricted to one word, the operand of an alphameric CODE subsequent entry may be identical to that allowed for an alphameric literal.

Consider the following example:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01		DA	3,,,0+INDEXWORD
02	STATE	CODE	14,,19@
03	ALABAMA		@ALA@
04	GEORGIA		@GA @
05	FLORIDA		@FLA@
06			

If the field, STATE, is tested during the running of the object program and the contents at that time are "ALA" the STATE will be considered to be ALABAMA; if the contents are "GA" the STATE will be considered to be GEORGIA; etc. If leading blanks appear, they may be omitted. Thus, if the values within the fields have been right-justified (rather than left-justified, as in the example above) the value for GEORGIA could have been written @GA@ rather than @GA @.

Processing Techniques

Since the order is irrelevant, CODE subsequent entries may appear in any sequence, provided that they follow a CODE header line. Only those code values which are to be tested need to be entered as a CODE subsequent entry, even when it is known that other data will appear in the field during the running of the object program.

The label of a CODE header line may be freely referenced in exactly the same ways that any other DA subsequent entry may be referenced; in each case the assigned location and the field control of its DA position will be compiled in the address portion of the referencing instruction. The data that will be operated on will be the data in the CODE field at that particular time in the running of the object program; a CODE subsequent entry will *not* cause a constant to be assigned to that area of storage. The names of the code values, which must be unique symbols and follow the usual rules for symbols, may not be referenced by symbolic machine instructions.

Additional Examples

Additional examples of the use of the CODE statement in conjunction with the LOGIC and DECOD macro-instructions appear on pages 173 and 177.

DC — Define Constant

The declarative statement, DC, may be used to enter numerical, alphameric and address constants (adcons) into the object program, and to assign names to constants for ease of reference. Like the DA statement, the DC statement consists of a header line and one or more subsequent entries with blank operation columns. Unlike the DA statement, a DC statement actually causes specified *data* to be compiled as a part of the object program. The DC header line causes the processor to assign storage locations to the constants which are defined by subsequent entries.

Source Program Format

DC Header Line

The formats for a DC header line are as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DC	+RDW
02	ANYLABEL	DC	−RDW
03	ANYLABEL	DC	RDW
04	ANYLABEL	DC	
05			

ANYLABEL may be any symbolic label or it may be omitted. It may *not* be an actual address. The entries DC and RDW must be written exactly as shown.

If plus or minus RDW is written in the operand, the processor will generate a single RDW with the indicated sign. If no sign is indicated, a minus RDW will be generated. This RDW, which will define the area containing all of the constants included in the subsequent entries, will be stored in the location immediately preceding the constant area. If present, the label of the DC header line may also be used to refer to the RDW. If the operand is blank, no RDW will be generated and the DC label will refer to the first word of the constant area.

DC Subsequent Entries

Four principal classes of constants may be defined in a DC subsequent entry: automatic-decimal numbers, floating-decimal numbers, address constants, and alphameric constants. These constants may be entered in either of two ways. (1) They may be entered separately, with individual symbolic names assigned to each, or (2) several constants of the same type may be defined by one subsequent entry, with one symbolic name assigned to the first location of the series.

Each constant may be entered in a separate core storage word or, within the limitations described below, several constants may be "packed" into the same word by means of field definers. Each subsequent entry may appear with or without a symbolic label.

In all examples of location assignments, it is assumed that the Autocoder assignment counter was at 1000 when the DC header line was encountered.

Automatic-Decimal Constants. This type of constant, used to enter signed numbers when the constant is to be referred to by a macro-instruction, may be from one to twenty digits in length. A decimal point may be included to indicate the magnitude of the number according to ordinary usage; it is neither stored in the constant nor saved with it, but merely serves as an indicator to Autocoder of the desired decimal-point placement. If the decimal point falls to the right of the right-most digit of the number, it may be omitted; the number will be considered an integer or an ordinary numerical constant.

If the constant is to be referred to by a symbolic machine instruction, however, it may neither exceed ten digits in length nor bridge words. The decimal point will be ignored; a symbolic machine instruction will treat the constant as an integer.

When used as a DC subsequent entry, automatic-decimal numbers may be individually named (with or without field definition) or written in a series, with the first number named.

INDIVIDUAL NAMES. The following example illustrates how automatic-decimal constants may be entered with individual symbolic names:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DC	
02	CONSTANT1		+12345.67890
03	CONSTANT2		-1.23
04	CONSTANT3		-41267936541.39217
05	CONSTANT4		-2
06			

The Autocoder processor would make the following actual location assignments:

Symbol	*Field Definition*	*Location*	*Contents*
CONSTANT1	0, 9	1000	+1234567890
CONSTANT2	0, 9	1001	−0000000123
CONSTANT3	0, 9	1002	−0000412679
	0, 9	1003	−3654139217
CONSTANT4	0, 9	1004	−0000000002

Note that each constant is right-justified in a separate storage word (two words in the case of a constant in excess of 10 digits). Reference to CONSTANT3 by a *symbolic machine instruction,* therefore, would result in the consideration of digit positions 0, 9 of location 1002 only.

The same constants could be packed into fewer words by using field definition in the subsequent entries, as in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DC	
02	CONSTANT1		+1234567890(0,9)
03	CONSTANT2		-1.23(0,2)
04	CONSTANT3		-41267936541.39217(3,18)
05	CONSTANT4		-2(9)
06			

Actual location assignments would then be as follows:

Symbol	*Field Definition*	*Location*	*Contents*
CONSTANT1	0, 9	1000	+1234567890
CONSTANT2	0, 2	1001	−1234126793
CONSTANT3	3, 9		
	0, 8	1002	−6541392172
CONSTANT4	9, 9		

When field definition is used with an automatic-decimal constant, it must define not less than the number of digits in the number and not more than 20 digits. Since the left field definition may be no greater than 9, the right field definition may be no greater than 28.

Constants written with field definition will be packed into words according to the following rules:

1. A change in sign will start a new word.
2. Field definition which would, in effect, result in overlapping in the same word will force the constant into a new word.

SERIES. The operand of a DC subsequent entry may include a series of signed automatic-decimal numbers, each without field control. All of the automatic-decimal numbers must share the same attributes, i.e., if the first number is written with a decimal point, they must *all* be written with a decimal point; if the first number has two decimal places, *all* must have two decimal places. (The number of integer positions, however, need not be the same.)

Since field definition is not allowed, each constant will be right-justified in one or two locations, depending on the length of the largest constant. Automatic-decimal constants 11-20 digits in length, therefore, should not be intermingled with constants 1-10 digits in length in the same subsequent entry since *every* constant would be assigned two locations.

The following entry will cause the indicated constants to be right-justified in separate words. (Note that commas are *not* used as separators.)

Line 3–5	Label 6–15	Operation 16–20	OPERAND 21 25 30 35 40 45 Basi 50
01	CONAREA	DC	
02			+15.2+3.8+9.7−2.1−30.4+1654.1
03			

Autocoder will make the following assignments:

Symbol	*Field Definition*	*Location*	*Contents*
CONAREA	0, 9	1000	+0000000152
		1001	+0000000038
		1002	+0000000097
		1003	−0000000021
		1004	−0000000304
		1005	+0000016541

Since the operand of the DC header line is blank, no RDW is generated. Therefore, when a symbolic machine instruction references the label CONAREA, it will refer to the first constant, +0000000152; CONAREA+1 will refer to +0000000038; CONAREA(8, 9)+2 will refer to +97; etc.

The following example illustrates the generation of a plus RDW to be associated with the constant area. Note also the use of integer constants.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	CONAREA	DC	+RDW
02			+152+38+97-21-304+16541
03			

Autocoder would make the following assignments:

Symbol	*Field Definition*	*Location*	*Contents*
CONAREA	0, 9	1000	+0010011006
		1001	+0000000152
		1002	+0000000038
		1003	+0000000097
		1004	−0000000021
		1005	−0000000304
		1006	+0000016541

The label CONAREA would now refer to the RDW defining the constant area; CONAREA(7, 9)+1 will refer to the first constant, +152; CONAREA+2 will refer to +0000000038; etc.

Floating-Decimal Numbers. Floating-decimal numbers may be entered in the operand of a DC subsequent entry with the format

$$\pm nF \pm m$$

where ±n is a decimal or integer number of not more than eight digits and ±m is a one- or two-digit exponent. If the sign preceding m is omitted, m is considered to be positive. A decimal point may be used to indicate implied positioning of the decimal point in the number ±n. If no decimal point appears, the number will be considered an integer. The value of the number is ±n multiplied by $10^{\pm m}$. Thus, −.98765432F+1 would represent the number $-.98765432 \times 10^1$. The exponent m may be omitted if equal to 0.

The Autocoder processor will consider the signs, the value of n, and the value of m of the entry in the format, ±nF±m. The processor will then generate a standard 7070 normalized floating-decimal word, which is of the form

$$\pm MMNNNNNNNN$$

where the sign is the sign of n. The number, ±nF±m, is normalized by placing n between +1 and −1 and by adjusting the value of m accordingly. The value of MM equals 50 plus or minus the adjusted value of m and the value of NNNNNNNN is the normalized value of n. For example, the number -1860.723×10^3 would be represented by −1860.723F+3. Autocoder converts this number to

the standard 7070 normalized floating-decimal format by normalizing the number to −.1860723F+7 and converting it to the form −571860723.

Since a floating-decimal number will always occupy ten digit positions and it may not bridge words, field definition is not allowed.

INDIVIDUAL NAMES. The following examples illustrate the use of individually named constants:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01		DC	
02	FLTNUMBER1		-3.4F
03	FLTNUMBER2		+34.567893F-2
04	FLTNUMBER3		-31.92F+7
05	FLTNUMBER4		-29567.1F-3
06	FLTNUMBER5		+12546F+15
07			

Assignment would be made as follows:

Symbol	*Field Definition*	*Location*	*Contents*
FLTNUMBER1	0, 9	1000	−5134000000
FLTNUMBER2	0, 9	1001	+5034567893
FLTNUMBER3	0, 9	1002	−5931920000
FLTNUMBER4	0, 9	1003	−5229567100
FLTNUMBER5	0, 9	1004	+7012546000

SERIES. The operands of DC subsequent entries may include a series of floating-decimal numbers, each written in the ±nF±m format described above. Unlike a series of automatic-decimal constants, the floating-decimal numbers *must* be separated by commas and the decimal point *need not* appear in the same place in each number.

The floating-decimal numbers used in the previous example could also have been written as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAN[D] 21 25 30 35 40	Autocoder— 65 70
01		DC		
02	FLTCON		-3.4F,+34.567893F-2,	+12546F+15
03				

The assigned locations would be the same as for the previous example with the exception that the single label FLTCON would be used to refer to the first constant at 1000, FLTCON+1 would refer to the second, etc.

Address Constants (Adcons). An adcon is used to produce an address constant which is the numerical representation (usually 4 digits) of the location

to which a given symbolic label has been assigned. As noted in a previous section, an adcon may be used as a literal or as a DC subsequent entry. When used as either, the form is ±SYMBOL. When used as a DC subsequent entry, adcons may be individually named (with or without field definition) or written in a series in which the first adcon may be named.

INDIVIDUAL NAMES. Adcons may be named individually; they may appear with or without field definition. Adcons written without field definition will be right-justified in separate words. Adcons written with field definition will be packed into words according to the following rules:

1. A change in sign will start a new word.
2. Field definition which would, in effect, result in overlapping in the same word will force the adcon into a new word.
3. If less than four digit positions are defined, the low-order digit(s) of the address will appear in the constant.
4. Adcons should not bridge words.

Consider the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DC	
02	ADCON1		-ADDRESS1
03	ADCON2		+ADDRESS2(0,1)
04	ADCON3		+ADDRESS3(2,5)
05	ADCON4		-ADDRESS4(6,9)
06			

Autocoder would first assign locations to ADDRESS1, ADDRESS2, ADDRESS3, and ADDRESS4. Assuming that these locations were 1125, 0025, 3542, and 6453, respectively, Autocoder would then assemble the adcons as follows:

Symbol	*Field Definition*	*Location*	*Contents*
ADCON1	0, 9	1000	−0000001125
ADCON2	0, 1	1001	+2535420000
ADCON3	2, 5		
ADCON4	6, 9	1002	−0000006453

SERIES. Several adcons may be written sequentially in a single DC subsequent entry as shown below. Field definition and address adjustment are not allowed; commas are not used between adcons.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DC	
02	ADCONS		-ADDR1+ADDR2+ADDR3-ADDR4
03			

Assuming that the labels have been assigned locations as in the previous example, Autocoder would assemble the adcons as follows:

Symbol	*Field Definition*	*Location*	*Contents*
ADCONS	0, 9	1000	−0000001125
		1001	+0000000025
		1002	+0000003542
		1003	−0000006453

The first adcon could be referred to by the label ADCONS; the second by ADCONS+1; etc.

Alphameric Constants. Alphameric information may be included in a constant area by using a DC subsequent entry similar to those in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–)
01		DC	
02	ANYLABEL		@12345@
03			@ CONSTANT ALPHAMERIC INFO @
04			@A @B @C @
05			@ABC@R
06			@@R
07			

ANYLABEL may be any symbolic label, or it may be omitted. Each operand must begin and end with the @ character. All characters between the *initial* @ character and the *final* @ character will be converted to double-digit form and assembled successively, beginning in the high-order positions of each location assigned. The sign of each word used to contain the characters will be alpha.

Packing of sequentially listed constants is automatic; it continues from line to line. If packing is to be avoided, therefore, blanks must be inserted within constants to fill out each storage word. If the processor finds that the total number of characters assigned through the end of a DC subsequent entry is not a multiple of five and the next entry is not an alphameric constant, blanks (double-digit 00) will be assembled in the low-order positions of the last word assigned.

The constant may contain the digits 0 through 9, the characters of the alphabet, blanks, and the special characters

. ⌑ & $ * - / , % # @

Note that the @ character may be included at any point in a constant field. It should also be noted that the processor will scan the *entire* operand for the *last* @ character. Therefore, this character *must not* appear in the remarks portion of the card if the constant field is to be properly defined.

A record mark may be defined by following the terminal @ with the character R, as explained below.

If symbolic machine instructions refer to the label of a DC header line which specifies the generation of an RDW, the assembled instruction will address the RDW. If, however, symbolic machine instructions refer to the label of a DC header line that does not specify RDW generation, or to the label of a DC subsequent entry defining an alphameric constant, they will act only on the first five characters of the area or the field, respectively. When defined for use by symbolic machine instructions, therefore, each subsequent entry should contain a maxi-

mum of five characters. A macro-instruction, however, may properly refer to the label of the header line or any subsequent entry and act on the entire constant area (exclusive of the RDW, if any) or field, regardless of length.

An alphameric constant may be used to define a message, as in the following example:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	EOFMSG	DC	-RDW
02			@REMOVE TAPE ON UNIT 00 @
03			@AND MOUNT NEXT @
04			

Autocoder will make the following assignments:

Symbol	*Field Definition*	*Location*	*Contents*
EOFMSG	0, 9	1000	-0010011008
		1001	R E M O V
		1002	E T A P
		1003	E O N
		1004	U N I T
		1005	0 0 A N
		1006	D M O U
		1007	N T N E
		1008	X T

Note that the label EOFMSG refers to the RDW defining the constant area.

In many instances, it may be desirable to name a portion of a long alphameric constant so that modification of the constant, such as entering variable tape addresses, codes, etc., may be readily accomplished. Thus, the constant in the previous example could have been defined as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	EOFMSG	DC	-RDW
02			@REMOVE TAPE ON UNIT @
03	TAPENO		@00@
04			@ AND MOUNT NEXT @
05			

Autocoder will make the following assignments:

Symbol	*Field Definition*	*Location*	*Contents*
EOFMSG	0, 9	1000	−0010011008
		1001	R E M O V
		1002	E T A P
		1003	E O N
		1004	U N I T
TAPENO	0, 3	1005	0 0 A N
		1006	D M O U
		1007	N T N E
		1008	X T

Note that Autocoder will assign the proper field definition to each symbolic name given so that reference to TAPENO will result in appropriate field control. If a field is to be referred to by symbolic machine instructions, however, the programmer is responsible for insuring that any named portion of an alphameric constant does not bridge words.

The following are examples of the use of the @ character *within* a constant field:

Line 3–5	Label 6–15	Operation 16–20	OPERAND 21–45
01		DC	
02	CONSTANT1		@A@B@C@
03	CONSTANT2		@@@
04			

Autocoder will make the following assignments:

Symbol	*Field Definition*	*Location*	*Contents*
CONSTANT1	0, 9	1000	A @ B @ C
CONSTANT2	0, 1	1001	@ b b b b

RECORD MARK. Since the record mark may not be entered through the IBM 7500 Card Reader, special provision is made for generating it in an alphameric constant by entering the letter R immediately following the terminal @ character.

If the last character generated by the current alphameric constant does not complete a word, the record mark will be assembled in positions 8 and 9 of that word, and any intervening positions will contain the double-digit representation of blanks. If the word is complete, the record mark will be assembled in positions 8 and 9 of the next word and positions 0 through 7 will contain the double-digit representation of blanks. Consider the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DC	
02	CONSTANT1		@ABC@
03	CONSTANT2		@DEFG@R
04	CONSTANT3		@HIJKL@R
05			

Autocoder would make the following assignments:

Symbol	*Field Definition*	*Location*	*Contents*
CONSTANT1	0, 5	1000	A B C D E
CONSTANT2	6, 9		
		1001	F G b b ‡
CONSTANT3		1002	H I J K L
		1003	b b b b ‡

(Note the packing of the first two characters of CONSTANT2 into the word which holds CONSTANT1.)

The following coding shows how a record mark may be entered and named so that field definers and/or address adjustment are not necessary when operating on it:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DC	
02	CONSTANT1		@ABCDE@
03	RECMARK1		@@R
04	CONSTANT2		@FG@
05	RECMARK2		@@R
06			

Autocoder will make the following assignments:

Symbol	*Field Definition*	*Location*	*Contents*
CONSTANT1	0, 9	1000	A B C D E
RECMARK1	8, 9	1001	b b b b ‡
CONSTANT2	0, 3	1002	F G b b ‡
RECMARK2	8, 9		

The first record mark may now be referred to by the label RECMARK1; the second by RECMARK2.

Additional Examples

The coding on the following page is furnished to illustrate various operands which may occur in a DC statement. Note that while various types of constants may be generated in the same DC area, they may *not* be mixed in the same subsequent entry.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–60) Basic Autocoder
01	ANYLABEL	DC	+RDW
02			+2
03			-123.4
04			+1.234(0.3)
05			+56.789(4.8)
06			-.1(9)
07		DC	-RDW
08			+17.2-200.1-1.7
09			-5.7F
10			+192.7F-2
11			-13F+8.+19.765F+3.-12.5F-2
12			+ADDRESS1
13		DC	RDW
14			-ADDRESS2(2,5)
15			-ADDRESS3(6,9)
16			-ADDRESS4+ADDRESS5-ADDRESS6
17			@123456789@
18		DC	
19			@A@R
20			@BLANKS AND SPECIAL CHARACTERS.)+$*-/,(#@@
21			@@R
22			

DLINE — Define Line

The DLINE statement and subsequent entries provide the programmer with a convenient means of specifying:

1. The layout of a print-line area.
2. The editing of fields to be inserted in that print-line area.

Various formats used in the subsequent entries provide the ability to define fields in which the following information may appear:

1. Constant data which is always to be included in the print line.
2. Alphameric data which is to be inserted by the object program.
3. Data edited to floating-decimal print format which is to be inserted by the object program.
4. Data edited to numerical print format which is to be inserted by the object program.

Several print-line formats are usually required for the preparation of a single report, e.g., a heading line, a transaction line, a total line, etc. These lines may be defined by means of separate DLINE header lines, each followed by its own subsequent entries.

The DLINE header lines set up and name the print areas; the subsequent entries name the printing positions to be used, define the characteristics of the fields, and cause the generation of constants which are to appear in the print area. The use of DLINE, therefore, results in the establishment of an output print image in core storage in which two core storage digit positions are reserved for each print position included. This area may initially include constant information specified by the programmer, e.g., record marks, captions, etc. Fields to be inserted by the object program will normally be moved into the image by an EDMOV statement. This macro-instruction will perform the necessary editing, insert decimal points and commas, convert data into double-digit form, etc. It is also possible to insert unedited information by means of the MOVE macro-instruction.

Source Program Format

DLINE Header Line

The basic format for the DLINE header line is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21 25 30 35 40 45)
01	ANYLABEL	DLINE	
02			

ANYLABEL is any symbolic label; it should not be omitted. It may *not* be specified as the operand of a DTF header line. DLINE must be written exactly as shown. The operand must be blank except for remarks.

DLINE Subsequent Entries

The header line must be followed by one or more subsequent entries, as in the example on the following page.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	LINENAME	DLINE	
02			10@TOTAL@
03			18@$@
04	GROSSAMT		19X,XXX.ZZ)DR,CR
05	CHECKAMT		60$X,XXZ.ZZ),C
06	ITEMNAME		80,94
07	FLVAR		95F
08			120@@R
09			

The DLINE entry and the subsequent entries, other than those which define constant fields, must be labeled. The operand of each subsequent entry must begin with a number which indicates the print position for the left-most character of the field. Thus, the word "TOTAL" in the example above would appear in print positions 10 through 14, the dollar sign on line 03 would appear in print position 18, etc. Print position 0 may *not* be specified.

The processor will issue a warning message if a print position beyond 135 is named or included in a DLINE area; however, the line may be of any desired length.

Descriptions and formats follow for each of the subsequent entries used to define the various types of data fields that may appear in a DLINE.

Constant Data. Constant information may be included in a DLINE area by using subsequent entries similar to those in the following examples:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	LINENAME	DLINE	
02	ANYLABEL		7@CONSTANT INFORMATION@
03			30@ BLANKS ARE INCLUDED @
04			100@$@
05			105@@@
06			109@A@B@C@D@
07			

A subsequent entry of this type may be written with a symbolic label, as in line 02 above. However, *unlike* the other types of fields which may be defined by subsequent entries, a constant field may appear *without* a label.

The operand must begin with the number of the print position for the left-most character in the field, immediately followed by the @ character. The @ character indicates that all *following* characters, up to (but not including) the *final* @ character, are to appear in the print area. The constant may contain numbers, characters of the alphabet, blanks, and any special characters (other than the record mark) acceptable to the input device used.

Note that the @ character may be included at *any* point in a constant field. Thus, line 05 would cause a single @ character to print in position 105 and line 06 would cause the following to print in positions 109-115:

A@B@C@D

A record mark may be entered by following the final @ with an "R" as shown in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	LINENAME	DLINE	
02			1@ABC@R
03			15@A@R
04			117@@R
05			

The record mark will be positioned according to the following rules:

1. When constant information appears between the initial and final @ character, the record mark will be placed in the next available print position ending in "0" or "5," with blanks inserted in the intervening print positions. The entry in line 02 above would cause a blank to appear in print position 4 and a record mark to appear in print position 5. The entry in line 03 above would cause blanks to appear in print positions 16 through 19 and a record mark to appear in print position 20.
2. The appearance of @@R will cause a *single* record mark to appear in the print position *named*. The entry in line 04 above would cause a record mark to appear in print position 117.

Constant fields, unlike the other fields which may be specified, will be initially loaded with the information specified. No provision is made for regenerating this information during the running of the program; if the object program includes coding to alter these constants, it must also include any coding necessary to restore them. Print positions for constant fields must appear in ascending sequence.

Alphameric Data Field. When data is to be inserted into an alphameric field in the print line, the field may be indicated by using subsequent entries similar to those in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	LINENAME	DLINE	
02	ANYLABEL1		1
03	ANYLABEL2		4 5
04	ANYLABEL3		21 100
05			

Any symbolic label may be used with an entry of this type; it should *not* be omitted, however. The operand must begin with the number of the print position for

the first or left-most character of the field. If more than one print position is to be included in the field, the first number must be followed immediately by a comma and the number of the print position for the last or right-most character in the field as in lines 03 and 04 above.

The object program will usually insert unedited alphameric data into a DLINE field of this type by means of a MOVE macro-instruction. If numerical data is to be edited to alphameric form, however, the EDMOV macro-instruction may be used. It is also possible to use symbolic machine instructions if the programmer is careful to consider word boundaries.

Data Edited To Floating-Decimal Print Format. When the object program uses the EDMOV macro-instruction to cause data to appear in the print area, edited to floating-decimal print format, the field in which the number is to appear must be defined by means of a subsequent entry under a DLINE statement. An entry of this type would be written as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	LINENAME	DLINE	
02	ANYLABEL1		13F
03	ANYLABEL2		101F
04			

Any symbolic label may be used with an entry of this type; the label should *not* be omitted, however. The operand consists of the number of the print position for the left-most character, followed immediately by the symbol F. A field of this type requires a total of thirteen print positions.

Floating-decimal numbers will be edited to the printing notation,

$$\pm MM\pm.NNNNNNNN$$

where $\pm MM$ is a two-digit exponent and $\pm .NNNNNNNN$ is a normal, eight-digit number (see page 57). The value of the number is $\pm .NNNNNNNN$ multiplied by $10^{\pm MM}$. For example, @9591999897@9695949372 (which in single-digit form would be -5198765432) will be printed as $\pm 01-.98765432$ representing the number $-.98765432 \times 10^1$.

Data Edited To Numerical Print Format. The combined use of DLINE and the EDMOV macro-instruction will provide a convenient way to do extensive editing of data to a numerical print format. This editing may include the insertion of the comma and decimal characters, the retention or elimination of leading zeros and commas, and the "floating" of a dollar sign. In addition, explanatory characters may be printed to identify plus or minus numbers.

A field of this type may be indicated by using subsequent entries similar to those in the following example:

Line 3–5	Label 6–15	Operation 16–20	OPERAND 21–50
01	LINENAME	DLINE	
02	ANYLABEL1		11X,XXX.ZZ
03	ANYLABEL2		19$XXX,XXX.XX
04	ANYLABEL3		30$XX,XXZ.ZZ)PLUS,MINUS
05	ANYLABEL4		45$ZZZ.ZZ)PLUS
06	ANYLABEL5		56$XXX,XXX,XXX,XXX,ZZ),MINUS
07	ANYLABEL6		80$XXX,XXX.XXX,XXX)PLUS,MINUS
08			

Any symbolic label may be used with an entry of this type; it should *not* be omitted, however. The operand must begin with the number of the print position for the first or leftmost character of the field. The rest of the operand is used to specify the desired format of the edited field, and (optionally) to specify the explanatory characters which are to identify plus or minus amounts.

Reference to a field of this type by an EDMOV macro-instruction will cause the generation of instructions which edit the data and insert dollar signs, commas, decimals, and explanatory characters. The decimal point in automatic-decimal variables will be aligned with the decimal point indicated in the DLINE field. The following characters may be used to indicate the format of the field:

Character	*Explanation*
X	This character represents a digit that is to be replaced by a blank if it is a high-order zero (i.e., a zero not preceded by a significant digit) to the left of the decimal point (if any). It will also be replaced by a blank, regardless of its position, if all digits are represented by Xs (as in lines 03 and 07 above) and all digits are zero.
Z	This character represents a digit that is always to be printed, even when it is a high-order zero.
$	This character represents a fixed dollar sign when followed by a Z (as in line 05 above) or a floating dollar sign when followed by one or more Xs (as in line 03 above). The $ need not appear (see line 02 above), but, when it is used, it must immediately follow the print position number. A floating dollar sign will print immediately to the left of the first printed digit. However, it will not float any farther to the right than the print position immediately to the left of the deci-

Character	Explanation
	mal point (if one appears). Thus, the \$ in line 03 might print in position 19, 20, 21, 23, 24, 25, or 26, depending on the placement of the leftmost significant digit in each amount printed. When all digit positions are replaced by blanks, the \$ will also be replaced by a blank. A fixed dollar sign will print in the print position indicated. The \$ in line 05 will always print in position 45, since the code character X does not appear to its right. (An illustration of another way to denote a fixed dollar sign appears on line 03, page 66, where a dollar sign is entered in a constant field.)
,	This character represents a comma when used *within* the format of the field. Commas may appear to the left or right of the decimal point. The comma will print only when digits are printed to its left. If the code character X allows the suppression of all high-order zeros to the left of the comma, the comma will also be suppressed. When the floating dollar sign is used, therefore, it will also float over commas that are to be suppressed. Since zeros to the right of the decimal point cannot be suppressed, a comma to the right of the decimal point will always appear.
.	This character represents the decimal point. Only one decimal point may appear in any given field. When all digits are replaced by blanks, the decimal will also be replaced by a blank.
)	A closing parenthesis may follow the rightmost position of the edited field. This code character does *not* occupy a print position; it is necessary only if explanatory characters follow, e.g., the PLUS and MINUS in lines 04, 05, 06, and 07, above. The use of explanatory characters will be explained below.

When the above characters are used to indicate the format of an edited field, at least one X or Z must appear, but not more than 20. When all digits are represented by an X and no significant digit appears in the amount, the entire field (including the dollar sign, commas, and decimal point positions) will be replaced by blanks.

When a field of this type is considered by the EDMOV macro generator, any explanatory characters which appear beyond the closing parenthesis (if this option is elected) are treated as follows:

1. Two consecutive blanks will signal the end of the operand.
2. Any characters which appear between the closing parenthesis and the comma to the right (or two consecutive blanks) will be inserted in the print line when the edited amount field is positive or blank.

   Any characters to the right of the comma will be inserted if the field is negative. The comma may be omitted if blanks are to be inserted for a negative field.
3. The explanatory characters may include numbers, letters of the alphabet, a *single* blank, and special characters (other than comma and record mark) acceptable to the input device used. A comma would signal the end of a set of characters; the maximum size for a set is one word.
4. The number of print positions required for explanatory characters will be equal to the largest of the two sets which may appear. Blanks will be inserted to the right of the shorter set when it appears.

Consider the following examples:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	LINENAME	DLINE	
02	FIELDONE		17$X,XXX.ZZ)+,−
03	FIELDTWO		30$X,XXX.ZZ)DR,CR
04	FIELDTHREE		50$X,XXX.ZZ)PLUS,MINUS
05	FIELDFOUR		70$X,XXX.ZZ),CR
06			

A positive and a negative amount, respectively, would print as follows:

FIELDONE, print positions 17 through 26.

$1,234.56+
$1,234.56−

FIELDTWO, print positions 30 through 40.

$1,234.56DR
$1,234.56CR

FIELDTHREE, print positions 50 through 63.

$1,234.56PLUS
$1,234.56MINUS

FIELDFOUR, print positions 70 through 80.

$1,234.56
$1,234.56CR

A *single* blank may appear at one or more points within a set of explanatory characters, as in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	LINENAME	DLINE	
02	FIELDNAME		27,$XXZ.ZZ +, −**
03			

A positive amount and a negative amount, respectively, would print as follows in print positions 27 through 38:

$123.45+
$123.45−**

Processing Techniques

General

At object program time, DLINE areas are initialized by condensed load cards which set all print positions within the area to blanks. These cards are followed by condensed load cards which initialize the constant data fields to the contents specified.

Since data fields will not necessarily occupy complete words, these cards will also cause blanks to be loaded into print positions which occupy the remainder (if any) of a word in which constant data falls. Constant data fields, therefore, should be entered in ascending order. If the processor encounters a constant data field which (1) is lower in sequence than the previous print position reserved, and (2) specifies a print position in a word in which a print position has been previously specified as constant data, the resultant condensed load card will overlay the previously specified print positions, thus yielding inconsistent results.

Although constant data fields are restricted to ascending sequence, the other subsequent entries may appear in any order. Thus, they may be used to specify print positions which cause different labels (and different characteristics, if desired) to be assigned to fields which occupy the same or overlapping print positions as in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	PRINTLINE	DLINE	
02	FIELDONE		20,$X,XXX.ZZ
03	FIELDA		20,28
04	FIELDTWO		30,$XXX.ZZ
05	FIELDB		30,$XXX.ZZ
06	FIELDTHREE		40,$X,XXX.ZZ
07	FIELDC		42,50
08			

It should be noted that a unique problem arises if the two fields overlap in the manner indicated by lines 06 and 07 above. Reference to either field will cause that field to be inserted as specified. However, if FIELDC is inserted, data may remain in print positions 40 and 41 from a previous use of FIELDTHREE and, when FIELDTHREE is inserted, data may remain in print positions 49 and 50 from a previous use of FIELDC. Thus, the object program must include additional coding to blank out data remaining from use of the other field.

Since all print positions are initially set to blanks, it is not necessary for the programmer to provide coding to blank out the print positions between fields. However, if the object program includes coding which alters these undefined print positions, it must also include coding to restore them to blanks when this becomes desirable.

If, in some instances, certain fields in a line are to be blank, they should be blanked out by a ZERO macro-instruction which references the label of the appropriate DLINE subsequent entry. If the ZERO macro-instruction refers to the label of a DLINE header line, however, it will permanently blank out constant data fields (if any) in the line as well as clear the variable fields. It will not, however, affect the dollar signs, commas, and decimals in the edited fields; these are inserted by coding which is generated by the EDMOV macro-instruction.

On-line Printing

During the running of the object program, data will be edited and moved into the DLINE area which serves as an image of the line to be printed. When all data is included in the area, the object program will issue an output statement which will transmit data from the DLINE area to the printer. This would normally be accomplished by a print command which refers to one or more RDWs defining the area(s) to be printed.

If the standard IBM 7400 printer panel wiring is used, line spacing and carriage spacing may be determined by digits 6, 7 and 9 of a Control Information Word as described in the 7070 Data Processing System Bulletin "IBM 7070 Utility Control Panels," form J28-6095, pages 8 and 9. Thus, the programmer is responsible for establishing a constant to fit the requirements for the Control Information Word as described in that bulletin. The print command must then reference one or more RDW(s) which causes the Control Information Word to be the first word sent to the printer, followed by the data in the DLINE area. Note that the Control Information Word will not be printed.

It should also be noted that the use of the standard IBM 7400 utility control panel provides for printing from typewheels 1 through 75 on one print cycle and from typewheels 76 through 120 on another print cycle. Thus, while a single DLINE statement may be used to describe a print line which uses both sets of typewheels, a single print command may *not* be used to print this entire line.

The following coding, however, would accomplish this purpose:

Line	Label	Operation	OPERAND
01		UW	2,FIRSTHALF
02			
03			
04		UW	2,SECONDHALF
05			
06			
07	FIRSTHALF	DRDW	+CIW1,CIW1
08		DRDW	-A,B
09	SECONDHALF	DRDW	+CIW2,CIW2
10		DRDW	-C,D
11			

The coding on line 07 above defines the first Control Information Word. Words 1-15 of the DLINE area are defined on line 08. The coding on line 09 defines the second Control Information Word. Words 16-24 of the DLINE area are defined on line 10.

The first Control Information Word (CIW1) would contain a digit "0" in position 5 to cause printing from typewheels 1 through 75. The second Control Information Word (CIW2) would contain a digit "1" in position 5 to cause printing from typewheels 76 through 120; a digit "6" in position 6 would cause spacing to be suppressed before printing.

Off-line Printing

When a tape is to be prepared for off-line printing, the following characteristics of the printer must be considered:

1. The first character of each line may be used to control carriage skipping and spacing.
2. The number of print-line records per tape block and the size of each record must not exceed that which the printer will accept.
3. A record mark may be required to separate the print lines within a tape block.

Thus, when the DLINE subsequent entries are written, the programmer should first refer to the appropriate printer manual to determine the allowable tape and printer formats.

When the first character of the print line is to be used to control carriage skipping and spacing, the other DLINE fields will print from the type wheel to the left of the position named. Therefore, if the character "1" on line 02 of the following example is used for carriage control, the characters "HEADER" will print in type wheels *1* through 6.

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	PRINTLINE	DLINE	
02	CARRIAGE		1@1@
03			2@HEADER@
04			

Thus, by defining the proper constant as "print position" 1 of each DLINE area the programmer can predetermine the carriage control for the various lines on the output listing; e.g., a header line would have a "1" in position 1, detail lines might have a "0" for double spacing, etc. Another method would be to insert the proper alphameric control character through programming, e.g., by reference to the label, CARRIAGE, above.

When the data has been edited and moved to the DLINE area, normal procedure would be to insert the assembled print line in an output file, where it is blocked under control of the Input/Output Control System. The following macro-instruction would accomplish this function:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01		PUT	PRINTLINE IN OUTPUTFILE
02			

The following coding might also be used:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01		PUT	OUTPUTFILE
02			
03			
04		MOVE	PRINTLINE TO OUTPUTFILE
05			

In each case, PRINTLINE is the label of the DLINE header and OUTPUTFILE is defined as a tape output file. PUT and MOVE are explained in the 7070 Data Processing System Bulletin, "IBM 7070 Input/Output Control System," form J28-6033-1.

As soon as the PUT (or PUT and MOVE) macro-instruction(s) have placed the DLINE area in the output file, the object program may proceed to prepare the next print line in the DLINE area.

Note that the DLINE area itself may *not* be defined as a file, i.e., the operand of a DTF header line may *not* be the label of a DLINE header line.

Particular consideration must be given to the use of the record mark in tapes prepared for off-line printing on the IBM 720. If more than one print-line record is to appear in a tape block, each record must end with a record mark, *with the exception of the last record in each block.* The object program, therefore, must include coding to count the lines that have been moved by the PUT macro-instruction and must cause a record mark to be inserted at the end of each record *except* the last one in a block. The end-of-job routine must also include coding to insure that no record mark appears at the end of the final print line.

Additional Examples

Additional examples of the use of the DLINE statement in conjunction with the EDMOV macro-instruction appear on pages 210 and 212.

DRDW—Define Record Definition Word

The declarative statement, DRDW, may be used to generate an RDW defining any area of storage specified by the programmer. It may also be used to cause the generation of one or more RDWs associated with an area defined by a DA or DC statement in some other part of storage, i.e., not immediately preceding the DA or DC area.

Source Program Format

Single RDW

When used to generate a single RDW for a given area, the format of the DRDW statement is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DRDW	+ADDRESS1,ADDRESS2
02	ANYLABEL	DRDW	−ADDRESS1,ADDRESS2
03	ANYLABEL	DRDW	ADDRESS1,ADDRESS2
04			

ANYLABEL may be any symbolic name or it may be omitted; it may *not* be an actual address. DRDW must be written exactly as shown. ADDRESS1 and ADDRESS2 are the limits of the area to be defined by the generated RDW. Either ADDRESS1 or ADDRESS2 (or both) may be an actual, *, or symbolic address. The * or symbolic addresses may appear with or without address adjustment. The sign of the generated RDW is determined by the sign preceding ADDRESS1; if a sign does not appear, however, the generated RDW will be signed minus. For example, if ADDRESS1 had been assigned to location 4372 and ADDRESS2 had been assigned to location 4408, the processor would have generated the RDW, −0043774400, at the point where the following DRDW was encountered:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DRDW	ADDRESS1+5,ADDRESS2−8
02			

The addresses in the operand of a DRDW statement may be the same, as in the following examples:

Line	Label	Operation	OPERAND
01	ANYLABEL1	DRDW	+*,*
02	ANYLABEL2	DRDW	-324,324
03	ANYLABEL3	DRDW	ADDRESS,ADDRESS
04			

When this is the case, the generated RDW will define a one-word area. Thus, line 02 above would cause the generation of the RDW, −0003240324.

Multiple RDWs

As explained under "DA-Define Area" and "DC-Define Constant," writing "RDW" on a DA or DC header line will cause the processor to generate an RDW(s) associated with the DA or DC entry and to assign it a storage location immediately preceding the defined area. Sometimes, however, it may be advantageous to cause the RDW(s) associated with a DA or DC statement to be generated in some other portion of storage, i.e., *not* immediately preceding the DA or DC area. This may be accomplished by using a DRDW statement with one of the following formats:

Line	Label	Operation	OPERAND
01	ANYLABEL	DRDW	+HEADRLABEL
02	ANYLABEL	DRDW	-HEADRLABEL
03	ANYLABEL	DRDW	HEADRLABEL
04			

ANYLABEL may be any symbolic name or it may be omitted; it may *not* be an actual address. DRDW must be written exactly as shown. HEADRLABEL must be the label of a DA or DC header line which appears as an entry in the program sequence. When this format is used, address adjustment is not allowed. If HEADRLABEL is the label of a DA header line, the number of RDWs generated will be the same as the number of areas designated by the DA header line. One RDW will be generated if HEADRLABEL is the label of a DC header line. A plus sign preceding HEADRLABEL will cause all RDWs generated to be signed plus; a minus sign will cause all RDWs generated to be signed minus. If no sign is shown, as in the format on line 03 above, all RDWs generated will be signed plus except the last which will be signed minus. When using symbolic machine statements, the programmer may make reference to the first RDW by referring to the label of the DRDW; address adjustment may be used to refer to the subsequent RDWs.

Additional Examples

The coding on the following page illustrates some of the operand forms which might appear in a DRDW statement.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	DRDW	HEADRLABEL
02		DRDW	+HEADRLABEL
03		DRDW	−HEADRLABEL
04		DRDW	+1,,99
05		DRDW	−325,,4999
06		DRDW	0,9989
07		DRDW	+*,*
08		DRDW	−*−1,*
09		DRDW	+*,*+2
10		DRDW	−*+1,,*+7
11		DRDW	*−1,*+2
12		DRDW	+ADDRESS1,,ADDRESS2
13		DRDW	−ADDRESS1+1,,ADDRESS2
14		DRDW	+ADDRESS1,,ADDRESS2+5
15		DRDW	−ADDRESS1−10,ADDRESS2−11
16		DRDW	ADDRESS1−2,,ADDRESS2+5
17		DRDW	+*,ADDRESS
18		DRDW	−ADDRESS,,*
19		DRDW	+ADDRESS+1,,*+1
20		DRDW	+0,*
21		DRDW	−*,4999
22		DRDW	+325,,ADDRESS+2
23		DRDW	−ADDRESS,9989
24		DRDW	0,ADDRESS
25		DRDW	*,ADDRESS
		DRDW	ADDRESS,9989

DSW — Define Switch

The primary function of the DSW declarative statement is to provide from one to ten digital switches which may be considered singly, or as a group, by the SETSW and LOGIC macro-instructions. SETSW and LOGIC will treat these switches as logically equivalent to electronic switches, although processed in a slightly different fashion. (The switches may *not* be referred to by electronic switch commands, e.g., ESN, ESF, etc.)

Each switch that is generated by the DSW statement occupies one digit position of a word and is considered OFF if its digit value is zero and ON if its digit value is other than zero, regardless of the sign of the word. Since the switches are generated at the point where the DSW statement is encountered, this statement should not appear within a series of machine instructions.

Source Program Format

The format for a DSW entry is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–50)	Basi
0 1	ANYLABEL	DSW	SWITCH1,-SWITCH2,+SWITCH3,etc.	
0 2				

ANYLABEL may be a symbolic name or it may be omitted. The entry DSW must be written exactly as shown. As many as ten symbolic switch names may appear, with a comma inserted between names. Continuation cards may be used, if necessary. The name of each switch must be unique; i.e., it must not be defined elsewhere as the label of another item. The initial setting of a switch is determined by the following:

1. If a plus sign, or no sign, precedes the name of a switch, the switch will be considered ON and set to "1."
2. If a minus sign precedes the name of a switch, the switch will be considered OFF and set to "0."

Processing Techniques

When the DSW statement is encountered, the processor will construct a one-word, positive, numerical field. The leftmost position will contain either "1" or "0," depending on whether the first-named switch in the DSW is to be initially ON or OFF. Succeeding digit positions will indicate the status of the remaining switches, in the order they are named. If less than ten switches are named, the remaining digits are set to zero. It should be noted that, while the switches are initially loaded as described, the programmer must provide additional coding (e.g., a SETSW macro-instruction) to reinitialize the switches if they are to be utilized in multi-pass programs.

Reference to the label of the DSW statement by symbolic machine instructions or

macro-instructions will result in reference to the *entire* word used to contain the switch settings. Consequently, if a label is supplied for a DSW, the entire set of switches may be tested or altered by LOGIC and SETSW statements.

The programmer is warned against trying to initialize electronic switches by using a DSW with an ORIGIN to 101, 102, or 103. The switches would be treated as digit switches, ignoring the fact that they are electronic switches.

Additional Examples

The coding on the following page illustrates various operands which might appear in a DSW statement. An additional example appears on page 187, where the defined switches are referred to by the SETSW macro-instruction.

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50 55 60 (Basic Autocoder →) 65 70 (Autocoder)
01		DSW	SWITCHA
02	ANYLABEL1	DSW	-SWITCHB
03		DSW	+SWITCHC
04	ANYLABEL2	DSW	SWITCHD,SWITCHE
05	ANYLABEL3	DSW	+SWITCHF,-SWITCHG,SWITCHH
06	ANYLABEL4	DSW	-SWITCH1,SWITCH2,SWITCH3,+SWITCH4,-SWITCH5,+SWITCH6,
07			SWITCH7,-SWITCH8,SWITCH9,-SWITCH10
08			

EQU — Equate

The EQU statement may be used to equate a symbol to:

1. An actual or symbolic address.
2. An index word or electronic switch number.
3. A channel, tape unit, combined channel and tape unit, combined arm and file, unit record synchronizer, inquiry synchronizer, or alteration switch number.

Thus, the EQU statement provides a convenient way to cause one or more symbols to be assigned to an actual location or to machine hardware. In this way, the same item may be referred to by different names in different parts of a program. Meaningful and easily remembered symbols may be used throughout the program, rather than the actual machine numbers which might be required in the operand of some Autocoder entries. In addition, when it is necessary to change the actual location or machine number, it is more convenient to change a single EQU statement than to alter each Autocoder statement which might otherwise contain the actual number.

Source Program Format

The general formats for an EQU statement are as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	EQU	ADDRESS
02	ANYLABEL	EQU	ADDRESS,Y
03	ANYLABEL	EQU	,Y
04			

The entry EQU must appear exactly as written. ANYLABEL is the symbolic name which is to be equated to ADDRESS. ANYLABEL may *not* be defined elsewhere in the same source program and may *not* be omitted.

ADDRESS may be any of the following:

1. An actual address, with or without field definition.
2. An index word or electronic switch number.
   address adjustment. This symbolic entry *must* appear as the label of an Autocoder entry elsewhere in the source program (not necessarily previously). It may *not* appear as the *label* of another EQU statement.
3. The number of an index word, with or without field definition.
4. The number of an electronic switch.
5. The number of a channel, tape unit, combined channel and tape unit, combined arm and file, unit record synchronizer, inquiry synchronizer, or alteration switch.
6. Omitted, in which case the first character of the operand must be a comma, followed by an X, S, T, etc., as described on page 86.

Y is replaced by a one- or two-character code which identifies ADDRESS as a particular piece of machine hardware.

Since EQU statements do not actually occupy core storage locations in the object program, they may be inserted at any point in the source program, provided that they are not intermingled with the subsequent entries under a DA, DC, or DLINE header line. (This is in contrast to the other declarative statements, which must be separated from the program instruction area.)

Processing Techniques

The method of coding each of the general uses of the EQU statement is described below.

Actual or Symbolic Address

A symbolic name may be made equivalent to an actual or symbolic address. The symbolic name to be equated is written in the label columns. The operand may contain an actual or symbolic address, with or without field definers. Address adjustment may also be used with a symbolic address.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DA	1
02	CUSTNO		2,,9
03		.	
04		.	
05		.	
06		.	
07		.	
08	CLASS	EQU	CUSTNO(0,,1)
09			

The above entries will cause the processor to assign the same location to the symbol CLASS as was previously assigned to CUSTNO. In addition, CLASS will be given field definers denoting the two high-order positions of CUSTNO, i.e., 2, 3.

EXAMPLE:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01		DA	1
02	RATE		2,,6A2,,3
03		.	
04		.	
05		.	
06		.	
07		.	
08	CLASS	EQU	RATE SEE WARNING BELOW
09			

The above entries will cause the processor to assign the same location to the symbolic name, CLASS, as was previously assigned to RATE. In addition, CLASS will be given the same field definers as RATE, i.e., 2, 6.

If address adjustment had been used in the preceding examples, only the assignment of a location would have been affected; the field definition would have been derived in the same manner as before. It is important to note, therefore, that the programmer is responsible for insuring that the field definition will actually be that desired for the new location.

The additional characteristics defined in the operand of a declarative statement will *not* be assigned to the name in the label field of an EQU statement. Thus, in the preceding example, CLASS will *not* be identified as a numerical field containing an automatic-decimal number. This creates no difficulty with symbolic machine instructions, since they do not use these characteristics, but limits the utility of the equated symbols in macro-instruction operands. When a symbol is equated to another symbol, a macro generator will treat the symbol in the label of the EQU statement as if it had the characteristics of a single whole word. Unless specifically desired, therefore, the use of equated symbols in a macro-instruction operand may cause program errors and should be avoided.

Two other methods are suggested for assigning two (or more) different symbolic names to the same field and the same characteristics. One method is to list both names as subsequent entries under the same DA, repeating the starting and ending digit positions and format indicators of the field, as in the example on page 45.

Another method of assigning two different symbolic names to the same location is to place the second name (with format indicators as desired) under a separate DA that is made equivalent to the first through the use of an Origin Control statement as explained on page 90.

Note that the EQU statement does not allow the following transitive relation:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
0 1	A	EQU	B
0 2		.	
0 3		.	
0 4	B	EQU	C
0 5			

Statements of this form are not acceptable; they will result in an incorrect location assignment. (The desired effect may, however, be obtained by writing: A EQU C; B EQU C.)

Index Word or Electronic Switch Number

A symbolic name may be made equivalent to an index word or electronic switch. The symbolic name to be equated is written in the label columns. The operand contains the one- or two-digit word (1-99) or electronic switch number (1-30), followed by a comma and the letter X or S, respectively. An index word may be field-defined as illustrated below. The index word or electronic switch that is equated to the symbolic name will be reserved during compilation; i.e., it will be passed over when Autocoder assigns symbolic index words and electronic switches to actual locations during Phase III.

EXAMPLES:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	LOOPCOUNT	EQU	1,X
02		.	
03		.	
04	IXWORD	EQU	52(2,5),X
05		.	
06		.	
07	SWITCHA	EQU	25,S
08			

The first entry will assign the name LOOPCOUNT to index word 1. The second entry will assign the name IXWORD to index word 52, with field definition (where applicable) of 2, 5. The third entry will assign the name SWITCHA to electronic switch 25.

Input/Output Units and Alteration Switches

A symbolic name may be made equivalent to a particular piece of machine hardware. The symbolic name to be equated is written in the label columns. The operand contains the number or value of the item, followed by a comma and an explanatory code character. The explanatory code characters used in EQU statement operands are as follows:

Item	*Code*
Tape Channel and Unit	CU
Tape Channel	C
Tape Unit	U
Disk Storage Arm and Unit	AF
Index Word	X
Electronic Switch	S
Alteration Switch	SN
Unit Record Latch	I
Unit Record Synchronizer	
Reader	R
Printer	W
Punch	P
Inquiry Synchronizer	Q
Typewriter	T

To illustrate, the following entry will cause the processor to assign the name RESTART to alteration switch 1:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	RESTART	EQU	1,SN
02			

This entry would make it possible for the programmer to write the more meaningful entry on line 01 below, rather than the entry on line 02:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01		BAS	RESTART,RESTORE
02		BAS	1,RESTORE
03			

This particular type of EQU statement may also be used for the specific purpose of identifying a particular item of hardware for the benefit of a macro generator (see "SNAP," example 1, page 228). When used for this purpose, the first entry in the operand may be omitted, as in the following examples:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	REGNAME	EQU	,X
02	SWITCH	EQU	,S
03	TYPEWRITER	EQU	,T
04			

Additional Examples

The coding on the following page illustrates operands which might appear in an EQU statement.

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	ANYLABEL	EQU	1000
02	ANYLABEL1	EQU	325(6,9)
03	ANYLABEL2	EQU	OTHERLABEL
04	ANYLABEL3	EQU	OTHERLABEL(0,4)
05	ANYLABEL4	EQU	OTHERLABEL+2
06	ANYLABEL5	EQU	OTHERLABEL(2,3)+1
07	XWORDNAME	EQU	1,X
08	XWORD	EQU	77(2,5),X
09	SWITCH	EQU	29,S
10	XWORDA	EQU	,X
11	SWITCHA	EQU	,S
12	CHANNEL	EQU	1,C
13	ACTIONTAPE	EQU	5,U
14	MASTERCU	EQU	21,CU
15	DISK	EQU	11,AF
16	ALTSWITCH	EQU	3,SN
17	UNITLATCH	EQU	2,I
18	READERNAME	EQU	1,R
19	PRINTNAME	EQU	2,W
20	PUNCHNAME	EQU	3,P
21	INQUIRY	EQU	1,Q
22	TYPEWRITER	EQU	,T
23			

Control Statements

Control statements are, in effect, orders to the processor which give the programmer control over portions of the assembly process. Thus, ORIGIN and LITORIGIN statements give the programmer control over the placement of his program in core storage. BRANCH statements cause the processor to produce execute cards containing unconditional Branches to locations specified by the programmer. An END statement will cause the processor to compile all remaining generated material and then produce an execute card containing an unconditional Branch to a location specified by the programmer. Control over the assignment of locations to symbolic index words and electronic switches is maintained through the use of XRESERVE, SRESERVE, XRELEASE and SRELEASE statements.

The formats and detailed descriptions of the use of these control statements are presented below. In all cases, the operation will be CNTRL. The labels must be prepared exactly as shown: ORIGIN, BRANCH, etc. The operand may vary as described for each control statement.

ORIGIN Control and LITORIGIN Control

ORIGIN statements order the processor to override its automatic assignment of storage locations and to begin the assignment of succeeding entries at the particular location specified by the programmer. Thus, they enable the programmer to control storage assignments of source-language input such as area definitions, constants, and instructions (including those generated "in-line" by macro-instructions). If an ORIGIN statement does not appear before the first such entry in a source program, the processor will begin the assignment of storage locations at an address specified to the Compiler Systems Tape. This address is originally 0325, but it may be altered as described in the IBM 7070/7074 Data Processing System Bulletin "IBM 7070/7074 Compiler Systems: Operating Procedure," form J28-6105.

LITORIGIN statements are used:

1. To partition or "segment" a program in order to enable the correct loading of a multi-phase program by causing the immediate compilation of all remaining material generated "out-of-line" in each segment (i.e., since the last previous LITORIGIN, or since the beginning of the program, if no LITORIGIN appears).
2. To regulate the placement of this material.

Material generated "out-of-line" in each segment includes generated constants, generated area definitions, generated symbolic subroutines, all literals, and all adcons. This material will normally be assigned locations immediately following the highest location assigned to the source program. The use of LITORIGIN, however, makes it possible to assemble this material at the end of each section or phase of a program so that it may be loaded with that section or phase. In addition, LITORIGIN is used to specify the beginning core-storage location at which the generated material is to be assigned.

The assignment of actual program locations is effected by means of location counters which may be named symbolically by the programmer and used by the Autocoder processor. The programmer has complete control over which counter is to be used while assigning locations to a given section of the program. In addition, he also controls the setting and resetting of the counters as desired. When in current use, however, a counter will be advanced automatically by the processor as locations are assigned; thus, after each assignment, it will always contain the address of the next location to be assigned. Provision is also made for "remembering" the minimum and maximum value attained by each counter. In certain cases, e.g., if the processor finds that addresses are to be assigned before the programmer has named the first symbolic counter, use is made of an internal (and, therefore, unnamed) counter.

Source Program Format

The ORIGIN and LITORIGIN statements have identical basic formats, written as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ORIGIN	CNTRL	NAMEONE, NAMETWO
02	ORIGIN	CNTRL	NAMEONE
03	LITORIGIN	CNTRL	NAMEONE, NAMETWO
04	LITORIGIN	CNTRL	NAMEONE
05			

The entries ORIGIN, LITORIGIN, and CNTRL must be written exactly as shown. NAMEONE is an entry which supplies the initial value to the location counter, NAMETWO.

NAMEONE may be any of the following types:

1. A symbolic label, with or without address adjustment, which has appeared previously.
2. The character *, with or without address adjustment. It will be considered to have the value (as adjusted) of the location counter in use at the time this statement is encountered.
3. An absolute machine address.
4. The symbolic name of a location counter which has been established in some previous ORIGIN or LITORIGIN statement, with or without address adjustment.
5. From 1 to 97 entries of the preceding types, separated by commas, enclosed by parentheses, and preceded by the characters MAX, as shown in the example on page 91. The largest (adjusted) value in this set will be used as the value to be established. If any of the entries in the set are the names of location counters, the *highest* value it has attained (rather than the last value) will be the value for comparison. Continuation cards, described on page 17, may be used as necessary.

In each of the five cases above, the value of NAMEONE will be placed in the counter NAMETWO, which will then be used to assign subsequent locations. If NAMETWO is omitted, an unnamed counter will be used for subsequent location assignments. NAMETWO may be the symbolic name of a previously defined location counter or the symbolic name of a new location counter. In either case, no address adjustment is allowed.

It is recommended that programs should normally be written with consistent NAMETWO usage: either (1) always spell out a NAMETWO counter, or (2) always omit any reference to NAMETWO. The latter case will mean that the unnamed counter is used throughout a program; this will often be suitable when a program is simple and straightforward, with little segmentation or overlaying of program areas. It can be seen that if NAMETWO is used consistently in a program, omitting a NAMETWO in some statement may cause the counter in use to be changed unwittingly.

When macro generators create an ORIGIN or LITORIGIN statement, there is an exception in its processing; i.e., an omitted NAMETWO will *not* cause the counter in use to be changed. The current counter will continue to be the one used for subsequent assignment because there is no way for a set of generated coding to refer to the counter that is in current use, but is obvious that this counter must remain the effective one.

To be compatible with Four-Tape Autocoder, an ORIGIN or LITORIGIN CNTRL with a blank operand has a special function. Each location counter, other than the counter named "S," is examined to determine the highest previous location (not necessarily the current value) assigned by *any* location counter. The value obtained is placed in the unnamed counter for subsequent assignment.

If statements are assigned locations in the index word area by means of an ORIGIN to an actual address, the corresponding index words will be reserved as they are encountered *during* the assignment pass. This will normally be done early in assembly to avoid duplicate assignment of these words.

Not more than 25 LITORIGIN statements may appear in one source-language program.

Processing Techniques

In its simplest form, the Autocoder ORIGIN statement is used to indicate the initial location which is to be used in assigning locations to a program. Suppose, therefore, that a program to be processed by Autocoder begins with the entry below and that no other ORIGIN or LITORIGIN statements are present:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ORIGIN	CNTRL	500,COUNTA
02			

In this case, Autocoder will establish a single location counter, COUNTA, with an initial value of 500. The entire program will be located in sequential locations beginning with 500, and all of the generated material will be assigned locations following the other programming entries.

The ORIGIN statement may also be used to:

1. Assign the same area of storage to several sections of a program.
2. Partition a program into several sections.
3. Assign program sections relative to the size and/or placement of other sections of the program.

Consider, therefore, a program that is to begin in location 1000. In this program, records that are read into a certain area of storage may have three different formats; therefore, three different sets of symbolic names and field definers may be desirable. The programmer may use separate DAS to define the three record formats that are to appear. Then, by proper use of ORIGIN statements, he may cause the processor to assign all three DAS to the same area of storage.

The ORIGIN statements for this program might appear as follows:

Line	Label	Operation	OPERAND
01	ORIGIN	CNTRL	1000,COUNTA
02		.	
03		.	
04		.	
05	FORMAT1	DA	10,,0+XWORD
06		.	
07		.	
08		.	
09	ORIGIN	CNTRL	FORMAT1,COUNTB
10	FORMAT2	DA	10,,0+XWORD
11		.	
12		.	
13		.	
14	ORIGIN	CNTRL	FORMAT1,COUNTC
15	FORMAT3	DA	10,,0+XWORD
16		.	
17		.	
18		.	
19	ORIGIN	CNTRL	MAX(COUNTA,COUNTB,COUNTC),COUNTD
20			

The last ORIGIN statement insures that succeeding entries (and/or generated material) will be assigned locations beyond the *longest* DA. If all three areas are the same length, or if the longest area is the last one to be defined, the last ORIGIN statement would not be necessary.

In the preceding examples, all generated material would have been assigned locations beyond the last program entry. However, if ORIGIN statements are used to cause subroutines or phases of a program to be assigned to the same or overlapping areas of storage, it may be desirable to include, in each section of the program, the generated material which it has produced. This may be accomplished through the proper use of the LITORIGIN statement as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ORIGIN	CNTRL	1000,COUNTA
02		.	
03		.	
04		.	(MAIN ROUTINE)
05		.	
06		.	
07	LITORIGIN	CNTRL	COUNTA
08	ORIGIN	CNTRL	COUNTA,COUNTB
09		.	
10		.	
11		.	(SUBROUTINE 1)
12		.	
13		.	
14	LITORIGIN	CNTRL	COUNTB
15	ORIGIN	CNTRL	COUNTA,COUNTC
16		.	
17		.	
18		.	(SUBROUTINE 2)
19		.	
20		.	
21	LITORIGIN	CNTRL	COUNTC
22	ORIGIN	CNTRL	COUNTA,COUNTD
23		.	
24		.	
25		.	(SUBROUTINE 3)
26		.	
27		.	
28	LITORIGIN	CNTRL	COUNTD

Each time the processor comes to a LITORIGIN statement, it assigns locations to all material generated "out-of-line" since the beginning of the program or since the last previous LITORIGIN statement. Since all remaining generated material would automatically be assigned to locations at the end of a program, it might have been possible to omit the final LITORIGIN statement.

The ORIGIN preceding subroutine 1 could have been written

07		.	
08	ORIGIN	CNTRL	*,COUNTB
09		.	

since COUNTA was being used when this statement was encountered.

If subroutines 2 and 3 were to start 25 words after the beginning of subroutine 1, the ORIGIN statements preceding them would have been written:

14		•	
15	ORIGIN	CNTRL	COUNTA+25,COUNTC
16		•	
17		•	
18	ORIGIN	CNTRL	COUNTA+25,COUNTD
19		•	
20			

If an ORIGIN statement to an *actual* address causes succeeding entries to be assigned locations in the index word area, the corresponding index words will be reserved and, if the entries are labeled, the index words will be named. The following example would cause index words 50 through 54 to be reserved and named and index word 55 to be reserved.

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	ORIGIN	CNTRL	50
02		DA	1
03	NAMEONE		0,9
04	NAMETWO		10,19
05	NAMETHREE		20,29
06	NAMEFOUR		30,39
07			40,49
08	ORIGIN	CNTRL	
09			

Since the index words are reserved only from the point at which the ORIGIN statement appears, this usage should normally appear at the beginning of the program or following a LITORIGIN statement.

Additional Examples

The coding on the following two pages illustrates some of the ways in which the assignment of locations may be manipulated through the use of ORIGIN and LITORIGIN control statements.

```
PAGE AA     PROGRAM CNTRL                                     7070 COMPILER SYSTEM VERSION OMYOB, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                                CDNO FD   LOC  INSTRUCTION       REF

01 1002   *  ILLUSTRATION OF ORIGIN AND LITORIGIN CONTROL STATEMENTS.
02 1003              NOP                           ASSIGNED BY UNNAMED COUNTER.  00001       0325 -0100090000
03 1004   ORIGIN     CNTRL 700                     UNNAMED COUNTER SET TO 700.
04 1005              NOP                           ASSIGNED BY UNNAMED COUNTER.  00002       0700 -0100090000
05 1006   0400       NOP                           ACTUAL ADDRESS IN LABEL.      00003       0400 -0100090000
06 1007              NOP                           ASSIGNED BY UNNAMED COUNTER.  00004       0701 -0100090000
07 1008   ORIGIN     CNTRL 800,COUNTER1            COUNTER1 SET TO 800.
08 1009              ZA1   +1                      ASSIGNED BY COUNTER1.         00005       0800 +1300000811
09 1010   LITORIGIN  CNTRL COUNTER1+10,COUNTER2    COUNTER2 ASSIGNS LITERAL.
                     LITERALS
10      X                  +1                                                    00006 00    0811 +1            0811
11 1011              ZA1   +2                      ASSIGNED BY COUNTER2.                     0812 +1300000923
12 1012   ORIGIN     CNTRL *+20,COUNTER2           COUNTER2 INCREASED BY 20.
13 1013   LABEL      ZA1   +2                      ASSIGNED BY COUNTER2.         00007       0833 +1300000923
14 1014   LITORIGIN  CNTRL LABEL+90,COUNTER3       COUNTER3 ASSIGNS LITERAL.
                     LITERALS
15      X                  +2                                                    00008 00    0923 +2            0923
16 1015              ZA1   +3                      ASSIGNED BY COUNTER3.                     0924 +1300001876
17 1016   ORIGIN     CNTRL COUNTER3-50,COUNTER4   *SEE NOTE BELOW.
18 1017              ZA1   +4                      ASSIGNED BY COUNTER4.         00009       0875 +1300111876
19 1018   LITORIGIN  CNTRL *+1000,S       SPECIAL COUNTER USED TO ASSIGN LITERALS.
                     LITERALS
20      X                  +3                                                    00010 00    1876 +3            1876
21      X                  +4                                                          11    1876 + 4           1876
22 1019              ZA1   +5             ASSIGNED BY SPECIAL COUNTER.                       1877 +1300000927
23 1020   ORIGIN     CNTRL COUNTER3,COUNTER3       *BACK TO COUNTER3, NOTE BELOW.
24 1021              ZA1   +5                      ASSIGNED BY COUNTER3.         00011       0925 +1300000927
25 1022   ORIGIN     CNTRL MAX(800,LABEL,COUNTER1,COUNTER2,COUNTER4,*),COUNTER5
26 1023              ZA1   +5                      ASSIGNED BY COUNTER5.                     0926 +1300000927
27 1024   LITORIGIN  CNTRL          SET UNNAMED COUNTER TO MAX ALL BUT S COUNTER.
                     LITERALS
28      X                  +5                                                          00    0927 +5            0927
29 1025              NOP                           ASSIGNED BY UNNAMED COUNTER.              0928 -0100090000
30 1026   ORIGIN     CNTRL *,COUNTER5     VALUE OF UNNAMED COUNTER IN COUNTER5
31 1027              ZA1   +5                      ASSIGNED BY COUNTER5.                     0929 +1300008000
32 1028   ORIGIN     CNTRL MAX(COUNTER5,S)         SET COUNTER5 TO MAX NAMED.
33 1029              ZA1   +5                      ASSIGNED BY COUNTER5.         00012       1878 +1300008000
34 1030   LITORIGIN  CNTRL 8000,COUNTER5           COUNTER5 ASSIGNS LITERAL.
                     LITERALS
35      X                  +5                                                    00013 00    8000 +5            8000
36 1031   *   NOTE THAT THE USE OF ADDRESS ADJUSTMENT IN LINE 17 DID NOT AFFECT
37 1032   *   COUNTER3
38 1033   *
```

ORIGIN COUNTER	INITIAL VALUE	LAST VALUE	HIGHEST VALUE	LOWEST VALUE
UNNAMED	0325	0928	0928	0325
COUNTER1	0800	0800	0800	0800
COUNTER2	0811	0833	0833	0811
COUNTER3	0923	0924	0924	0923
COUNTER4	0875	0875	0875	0875
COUNTER5	0926	1878	1878	0926
S	1876	1877	1877	1876

BRANCH Control

The BRANCH statement will cause the processor to produce an execute card containing an unconditional Branch instruction. When encountered during the loading of the object program, this instruction will cause the normal loading process to stop and a branch to be executed to a specified location.

Source Program Format

The BRANCH statement should be written as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	BRANCH	CNTRL	ADDRESS
02			

BRANCH and CNTRL must be written exactly as shown. ADDRESS is the actual, *, or symbolic address to which the branch is to be made after the preceding portion of the program has been loaded into storage. If an * or symbolic address is used, address adjustment and/or indexing may be specified. If an actual address is used, indexing *only* may be specified.

Processing Techniques

A BRANCH statement may be used in conjunction with an ORIGIN statement to execute portions of a program already loaded into storage and to overlap these with other instructions, as in the following example:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	START		(FIRST INSTRUCTION)
02		.	
03		.	
04		.	
05		.	
06	XYZ	B	LOADPROG
07	BRANCH	CNTRL	START
08	ORIGIN	CNTRL	START, COUNTER
09			

When the resultant object program is being loaded, the loading operation will be interrupted by a branch to the location assigned to the symbol START, followed by the execution of the instructions from START through the instruction located at XYZ. The instruction located at XYZ will cause a branch back to the load program, which will then resume the loading of the remainder of the object program. Since the following entries were assigned locations beginning again with the location START, they will overlap the instructions which have already been executed. (In this example, it is assumed that the starting location of the load program is symbolic location LOADPROG.)

Additional Examples

The following coding illustrates various operands which might appear in a BRANCH statement:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	BRANCH	CNTRL	*
02	BRANCH	CNTRL	*-2
03	BRANCH	CNTRL	*+1+X1
04	BRANCH	CNTRL	*+XWORD
05	BRANCH	CNTRL	325
06	BRANCH	CNTRL	0+X1
07	BRANCH	CNTRL	0+XWORD
08	BRANCH	CNTRL	LABEL
09	BRANCH	CNTRL	LABEL+10
10	BRANCH	CNTRL	LABEL+X84
11	BRANCH	CNTRL	LABEL+2+XWORD
12			

END Control

The END statement is an indication to the processor that the end of the source-language program has been reached. When this statement is encountered, the processor will assign locations to all material generated since the last previous LITORIGIN statement or since the beginning of the program if LITORIGIN was not used. In addition, it then produces an execute card containing an unconditional Branch instruction which, when encountered at the end of the loading of the object program, will cause a branch to a specified location.

Source Program Format

The END statement may be written as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	END	CNTRL	ADDRESS
02	END	CNTRL	
03			

END and CNTRL must be written exactly as shown. ADDRESS is the symbolic, actual, or * location to which the branch is to be made after the entire object program has been loaded into storage. It should be noted that a BRANCH statement may not ordinarily be used for this purpose since it would cause the branch to occur before the generated material (if any) was loaded. If an * or symbolic address is used, address adjustment and/or indexing may be specified. If an actual address is used, indexing *only* may be specified.

Processing Techniques

If the END statement is used, it must be the last entry in the program. If it is not used, or if it is used with a blank operand, the processor will assign locations to all generated material and then generate an unconditional branch to an address specified to the Compiler Systems Tape. This address is originally 0325, but it may be altered as described in the 7070/7074 Data Processing System Bulletin "IBM 7070/7074 Compiler Systems: Operating Procedure," form J28-6105.

Additional Examples

The coding on the following page illustrates various operands which might appear in an END statement.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	END	CNTRL	*
02	END	CNTRL	*+10
03	END	CNTRL	*-5+XWORD
04	END	CNTRL	*+X15
05	END	CNTRL	325
06	END	CNTRL	0+XWORD
07	END	CNTRL	0+X11
08	END	CNTRL	LABEL
09	END	CNTRL	LABEL-10
10	END	CNTRL	LABEL+XWORD
11	END	CNTRL	LABEL+5+X45
12	END	CNTRL	
13			

XRESERVE Control and SRESERVE Control

References to the actual address of an index word or electronic switch may be made at the discretion of the programmer. Normally, however, he will use symbolic names which the compiler will then assign to actual addresses. XRESERVE and SRESERVE statements cause the processor to reserve index words and electronic switches, respectively, so that they will be "unavailable" when this assignment is made; i.e., symbolic names will not be assigned to the address(es) reserved.

Source Program Format

The XRESERVE and SRESERVE statements may be written as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	XRESERVE	CNTRL	N_1, N_2-N_3, etc.
02	XRESERVE	CNTRL	
03		.	
04	SRESERVE	CNTRL	N_1, N_2-N_3, etc.
05	SRESERVE	CNTRL	
06			

The entries XRESERVE, SRESERVE and CNTRL must be written exactly as shown.

The operand field contains the one- or two-digit number of the index word(s) or electronic switch(es) to be reserved, or is blank. Continuation cards, described on page 17, may be used as necessary. The entry N_2-N_3 reserves all of the index words or switches between and including N_2 and N_3. When this form is used, N_3 must be a number *greater* than N_2. The two forms of operand entries may be used exclusively or intermixed freely on each card. A blank operand will cause all of the index words or electronic switches to be reserved.

Processing Techniques

As described on page 19, the XRESERVE and SRESERVE statements provide one of several methods of making index words or electronic switches unavailable for assignment to symbolic names. Unlike some of the other methods described, XRESERVE and SRESERVE statements affect the availability table only when they are encountered in the statement-by-statement processing during the Phase III pass which assigns index word and electronic switch addresses.

XRESERVE and SRESERVE statements are usually placed at the beginning of a source program. However, they may also appear at a later point(s) in the program. When this is the case, it is possible that a previous symbolic index word or electronic switch name may already have been assigned to the address specified. The programmer must therefore be cautious in the use and placement of these statements.

Additional Examples

The following coding illustrates various operands which might appear in an XRESERVE statement:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	XRESERVE	CNTRL	1
02	XRESERVE	CNTRL	90
03	XRESERVE	CNTRL	3,4,79,2
04	XRESERVE	CNTRL	9-21
05	XRESERVE	CNTRL	1-4,7-8,79-80,45-47
06	XRESERVE	CNTRL	1,3-4,7,9,11,15,13,17-19,21,
07			23,25,27,29,31-34,37-39,50
08	XRESERVE	CNTRL	
09			

The following coding illustrates various operands which might appear in an SRESERVE statement:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	SRESERVE	CNTRL	2
02	SRESERVE	CNTRL	29
03	SRESERVE	CNTRL	3,4,15,7
04	SRESERVE	CNTRL	9-21
05	SRESERVE	CNTRL	1-4,7-8
06	SRESERVE	CNTRL	1,3-5,7,9,11,13,15,
07			21,25-27
08	SRESERVE	CNTRL	
09			

XRELEASE Control and SRELEASE Control

Through the use of XRELEASE and SRELEASE statements, it is possible to make index words or electronic switches, respectively, available for later assignment, even though they may have been reserved through some previous assignment or listed as unavailable in the initial availability table. Thus, even though a given section or phase of a program may make extensive use of symbolic index words and switches, the use of the XRELEASE and SRELEASE statements will cause the processor to re-establish the availability of these index words and electronic switches for later assignment.

Source Program Format

The XRELEASE and SRELEASE statements may be written as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–)
01	XRELEASE	CNTRL	N_1,N_2–N_3,XWORDA,XWORDB,etc.
02	XRELEASE	CNTRL	
03			
04			
05	SRELEASE	CNTRL	N_1,N_2–N_3,SWITCHA,SWITCHB,etc.
06	SRELEASE	CNTRL	
07			

The entries XRELEASE, SRELEASE, and CNTRL must be written exactly as shown. The operand field contains the symbolic name or one- or two-digit number of the index word(s) or electronic switch(es) to be released. Continuation cards, described on page 17, may be used as necessary. The entry N_2-N_3 releases all of the index words or switches between and including N_2 and N_3. When this form is used, N_3 must be a number *greater* than N_2. The various operand entries may be used exclusively or intermixed freely for each statement. A blank operand will cause all of the index words or electronic switches to be released.

Processing Techniques

As noted on page 19, the processor uses an availability table to determine the assignment of symbolic names to actual index words and electronic switches. This availability table differentiates between the index words and electronic switches which have not yet been assigned in a given program, and those which have been assigned but subsequently released. A released index word or electronic switch is not reassigned until all others have been assigned for the first time.

A given symbol is *always* assigned to the same index word or electronic switch, even if that particular index word or switch is subsequently released for possible later assignment to another symbolic name.

Since the XRELEASE and SRELEASE statements are designed to allow the assignment of more than one symbolic name to the same location, it is imperative that the programmer use these entries with great care. Otherwise, conflicting use of the same index word or the same electronic switch may result in inconsistent program

results. These statements should be used only where the programmer is certain that the index word or electronic switch is no longer required in its previous role, e.g., at the end of a given phase in a multi-phase program, perhaps following a LITORIGIN statement. It should be noted that where more than one name is made equivalent to a given index word or electronic switch, an XRELEASE or SRELEASE statement referring to any one of the names (or to the actual address itself) has the effect of a statement referring to *all* of the equivalent names.

Additional Examples

The coding on the following page illustrates various operands which might appear in XRELEASE and SRELEASE statements.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–60) Basic Autocoder	(61–75) Autocoder
01	XRELEASE	CNTRL	1	
02	XRELEASE	CNTRL	90	
03	XRELEASE	CNTRL	3,4,79,2	
04	XRELEASE	CNTRL	9-21	
05	XRELEASE	CNTRL	1-4,XWORDA	
06	XRELEASE	CNTRL	XWORDB,XWORDC	
07	XRELEASE	CNTRL	XWORDD,10-20,30-35	
08	XRELEASE	CNTRL	XWORDE,6-8,XWORDF,79,10-12	
09	XRELEASE	CNTRL	1,10,XWORDNAMEA,11-13,XWORDNAMEB,XWORDNA	MEI,XWORDNAMED,
10			XWORDNAMEE,XWORDNAMEF,XWORDNAMEH,25,XWOR	DNAMEG,29-31,
11			XWORDNAMEL	
12	XRELEASE	CNTRL	XWORDA	
13				
14	SRELEASE	CNTRL	2	
15	SRELEASE	CNTRL	29	
16	SRELEASE	CNTRL	3,4,15,7	
17	SRELEASE	CNTRL	9-21	
18	SRELEASE	CNTRL	1-4,SWITCHA	
19	SRELEASE	CNTRL	SWITCHB,SWITCHC	
20	SRELEASE	CNTRL	SWITCHD,10-20,27-29	
21	SRELEASE	CNTRL	SWITCHE,2-7,SWITCHF,29,11-15	
22	SRELEASE	CNTRL	1,3,SWITCHI,SWITCHF,7-11,SWITCHC,SWITCHD	,SWITCHE,20,
23			SWITCHM,SWITCHG,SWITCHH,SWITCHA,SWITCHB,	SWITCHK,
24			SWITCHL,29	
25	SRELEASE	CNTRL	SWITCHM	

Imperative Statements

Autocoder imperative statements include low-level statements, called "symbolic machine instructions," which are very much like the 7070 machine language operation codes, and high-level statements, called "macro-instructions," which bear no resemblance to machine language. While the symbolic machine instructions offer flexibility and control over each detail of the coding, the macro-instructions provide a more powerful and convenient way to state a problem. The macro-instructions usually produce a number of machine language instructions in the object program.

Each macro-instruction and symbolic machine instruction has a unique mnemonic operation code consisting of from one to five alphameric characters. For example, the Autocoder equivalents of the Branch and Make Sign Plus machine commands are B and MSP, while for the Compare and Set Switch macro-instructions, COMP and SETSW are used.

Symbolic Machine Instructions

When a symbolic machine instruction is encountered in a source program by the 7070 Autocoder processor, it is converted into a 7070 machine language command. The symbolic machine instructions are sometimes referred to as "one-for-one" instructions since each unique mnemonic representation will cause one 7070 operation to be inserted in the object program. For example, the Branch, Lookup Lowest, and Index Word Load and Interchange commands have the corresponding Autocoder mnemonics B, LL, and XLIN. Each time the 7070 Autocoder processor encounters these mnemonics in the source program it will insert the machine language operation code +01, +66 and −48, respectively, into the object program. An alphabetic list of all the Autocoder symbolic machine instructions, indicating what is permissible in the operand field and the order in which the information must be entered on the coding form, is contained in Appendix D.

The correct order of entry for operand parameters of symbolic machine instructions is the operand address followed by field definers, address adjustment, and indexing, as illustrated by the following examples:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01		CD	FIELDA(2)+4+IWORD,6
02		.	
03		.	
04		XA	IWORD,FIELDA+4+IWORD
05			

Assume that FIELDA has been defined as word 2000 and that the indexing portion of index word IWORD contains 0100. In the first example, digit position 2 of location 2104 will be compared to a "6." In the second example, 2104, considered plus, will be algebraically added to the number 0100 in the indexing portion of IWORD.

The IBM Reference Manual "7070 Data Processing System," form A22-7003-2, contains numerous illustrations of how symbolic machine instructions are written and gives examples of the machine language instructions which will be assembled from them.

The following is a section of a payroll routine which further illustrates the use of symbolic machine instructions. Note that remarks may appear anywhere in the operand, provided two blank spaces separate the remarks from the operand of the instruction.

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21 25 30 35 40 45)
01	CALCTAX	ZA3	TAXCLASS DETERMINE IF
02		M	-1300 PAY IS
03		A2	GROSS TAXABLE
04		BM2	NOTAX BRANCH IF NO TAX
05		ZA3	9992
06		M	+18 CALCULATE TAX
07		SRR	2
08		B	*+2
09	NOTAX	S2	9992 CLEAR ACCUM2
10		ST2	WITHHOLD STORE TAX AMOUNT
11			

The use of macro-instructions described, below, will make possible the coding of a source program with only limited use of symbolic machine instructions.

Macro-Instructions

A macro-instruction represents a single operation entry on the Autocoder coding sheet that is converted during assembly into a sequence of machine instructions. Autocoder macro-instructions, including those used for input/output operations, free the programmer to a large extent from attention to machine details such as location assignment and data flow in carrying out the operations required by frequently encountered problems. The user may extend Autocoder by adding appropriate macro generators to the system to process newly-created macro-instructions.

The macro-instructions fall into several categories, as follows:

Category	*Macro-Instructions*
Input/Output:	OPEN, GET, PUT, PUTX, CLOSE, END
Arithmetic:	ARITH
Decision Making:	COMP, CYCLE and RECYC, DECOD, LOGIC, ZSIGN
Initialization:	SETSW, ZERO, FILL
Data Movement:	EDMOV, MOVE, SHIFT
Reference:	SNAP

A full discussion of the input/output macro-instructions can be found in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," form J28-6033-1. The programmer should not attempt to use these macro-instructions without careful study of the material contained in that bulletin, especially the DIOCS, DTF, and DUF entries. For convenient reference, however, the formats and brief descriptions of six of the principal input/output macro-instructions are included in this manual.

Programming features common to all macro-instructions are outlined immediately below. A detailed description and the format of each individual macro-instruction follow. Examples are included showing typical source-program statements and the coding generated from them by the macro generators.

Label. A macro-instruction which will be referred to elsewhere in the program is written with a label. In all other cases, the label column is blank. The label may be any symbolic label acceptable for a symbolic machine instruction, i.e., beginning with a letter, possibly followed by any combination of up to nine letters or numbers (no special characters). This label will reference the first instruction generated from the macro-instruction.

Certain labels are forbidden. The following sets of characters have special meanings when used in the operands of the macro-instructions specified. They should not, therefore, be simultaneously employed as labels in programs using these macro-instructions:

Characters	*Macro-Instructions*
ABS	ARITH, COMP
AND	LOGIC, FILL
E	LOGIC
G	LOGIC
IN	PUT, PUTX
L	LOGIC, SHIFT
LC	SHIFT
LS	SHIFT
NOT	LOGIC
NOZERO	ZSIGN
OR	LOGIC
R	SHIFT
RR	SHIFT
RS	SHIFT
TO	EDMOV, GET, MOVE
WITH	FILL
The name of any function in the Macro Table	ARITH

Actual Addresses. The operand may not include actual addresses, except where specifically allowed by individual macro-instructions.

Field Characteristics. When a program is assembled, it is not known what the contents of the various fields will be at object program time. Therefore, all macro generators must proceed on the assumption that these contents will conform to the characteristics of the field as defined by some declarative statement. If a field has been defined by a DA subsequent entry as alphameric, instructions will be generated to treat the contents as if they were alphameric, and difficulties may arise if, during the object program, numerical information has been stored instead. The same is true if alphameric information is put into numerical fields, or if the numerical modes are not distinguished from each other.

Asterisks(*). When a macro-instruction is written, the programmer does not know how many machine instructions will be generated in its place; therefore,

addresses using the asterisk symbol are not acceptable. For example, if a macro-instruction must reference the next following instruction of the source program, this should be done by means of a label attached to that instruction, not by the address * + 1. The same considerations make it impossible to do address arithmetic on a macro-instruction label or on any other label by amounts that would carry the address across a macro-instruction.

Alphameric Literals. Alphameric literals may appear in macro-instruction operands, provided they are meaningful to the specific instruction involved. It is not possible, however, to write an alphameric literal that includes @ as one of its characters. The scan for macro-instructions, sensing an operand like @AAAAA@ AAAA@, would take the second @ to signal the termination of the literal. If such a constant is required by the nature of the program, it must be entered as a subsequent entry under a DC header line; it may then be referenced in the operand of the macro-instruction by its symbolic name.

Address Modification. Although field definition is not permitted within macro-instruction operands (except through the device of referencing a label of a DA, DC, or DLINE subsequent entry specifying field definition), indexing and address adjustment are permitted. The conventions for writing these, however, differ from those applicable to symbolic machine instructions. In particular, indexing *precedes* address adjustment if both are present.

Indexing may be specified by referencing the symbolic name of index words or their actual one- or two-digit number. When actual indexing is used, the number of the index word may not be signed, nor should it be preceded by an X. Address adjustment must be signed plus or minus; it is this which distinguishes one- or two-digit adjustments from indexing.

Address modifiers, both index words and address adjustment, must be enclosed by parentheses. The left parenthesis should be in the column immediately following the last character of the address being modified. No blanks should occur anywhere within the entire address modification. If both indexing and address adjustment are used, indexing *precedes* address adjustment, and one set of parentheses should enclose them both.

The following are acceptable examples of modification for macro-instruction operands:

Operand	*Explanation*
TABLE(ROW)	The symbolic address TABLE is indexed by the index word ROW.
LIST(+34)	The symbolic address LIST is incremented by 34.
LIST(34)	The symbolic address LIST is indexed by index word 34.
CHART(LINE−10)	The symbolic address CHART is decremented by 10; then the address is indexed by the index word LINE.
ARRAY(29+17)	The symbolic address ARRAY is incremented by 17; then the address is indexed by the index word 29.

Continuation Cards. Since macro-instructions frequently have long operands, it will often be necessary to use continuation cards to accommodate all the required parameters. All macro-instructions are limited to five cards, i.e., four continuation cards in addition to the header card.

On a continuation card the label and operation columns must be blank, and the continuation of the operand portion must begin in column 21; i.e., it must be left-justified in the operand column of the coding sheet. The entire operand, columns 21-75, may be used.

In the header card or continuation cards (other than the last), each operand need not extend across the entire operand column; it may end with the comma following any parameter and continue on the succeeding card.

In two cases, however, the operand can not be broken off in the middle to be distributed over two cards. These exceptions are:

1. *Alphameric Literals.* Alphameric literals may be up to 120 characters in length and may, in consequence, have to be distributed over two or more cards. This is permitted, provided that the operand columns of all cards but the last are filled to the very end, i.e., through column 75, and that the continuation begins in column 21 of the next card. Blanks which are to be included in the alphameric literal are regarded as part of the literal and may appear in *any* column, including column 75 and column 21. Any extraneous blanks or remarks which appear at the right end of any card before the terminal @ character will be regarded as part of the literal and these characters will be incorporated into the literal.
2. *Address Modification.* When a parameter is modified by indexing and/or address adjustment, separation at the end of a card may be made as follows: The name of the parameter is placed on the first card. The address modification which follows may be on the same card or it may be broken off at any point following the left parenthesis (except *within* the actual or symbolic addresses involved) and continued on the next card beginning in column 21. If the last character of a parameter falls in column 75, then the left parenthesis may be placed in column 21 of the next card.

The examples on the following page illustrate the method of separating alphameric literals and address-modified parameters on continuation cards. Examples 1 and 2 illustrate alphameric literals; examples 3-7 illustrate address modification.

Punctuation and Spacing. In general, each macro statement has its own conventions of punctuation, which are stated in detail in the descriptions of the individual instructions. Illegal characters or other faulty punctuation will be regarded as an error condition. For all macro-instructions, any entry in an operand portion of the coding sheets that is preceded by two blanks will not be processed, unless the blanks are inside an alphameric literal. As a general rule, it is recommended that no blanks be written, especially following commas, equal signs, or other punctuation marks, unless a specific demand is made under "Source Program Format" in the description of the individual macro-instruction (e.g., in the MOVE statement on either side of the operator TO) or when blanks appear in address modification as a result of the use of continuation cards.

Remarks. Remark entries may be made at the end of the operand portion of the coding sheet just as with symbolic machine instructions. At least two blanks must precede the first character of a remark entry. These entries, which may include the @ character, will be listed but not processed. If a macro-instruction requires continuation cards, remarks are not necessarily confined to the last card; parameter entires, except for alphameric literals, may be terminated wherever the programmer desires, subject to the rules stated under "Continuation Cards," and continued on the next card, leaving room for remarks at the right end. Attention is called to the fact that macro-instructions are limited to a total of five cards for each instruction; if an operand is very long, remark entries on the header or con-

tinuation cards may waste needed space. In such a case, remarks may be entered on separate comments cards immediately preceding or following those containing the macro-instruction proper.

Error Conditions. If the macro generator detects an error condition in analyzing the source statement, an error or warning message will be issued. Warning messages inform the programmer that the machine instructions being generated may have certain unintended effects in the object program, such as accumulator overflow in ARITH, branching to the same location regardless of the outcome of the test in ZSIGN, etc. The programmer should recheck his use of the macro-instruction to make sure that he has employed it correctly and that the special condition will either not arise in his program or that it is intentional. Error messages are issued if a programming error has made it impossible for the macro generator to generate meaningful instructions on the basis of the source statement; in such a case, a NOP will be generated to aid in patching. Assembly will not be interrupted.

Frequently a macro generator will pass a portion of its work on to another generator by putting out what is called a "lower-level" macro-instruction. This will be automatically assembled by the processor, with all generated instructions properly sequenced. The possibility exists, however, that a parameter passed to a lower-level macro generator for processing could bring about an error condition in that generator. In that case, the error message issued would be one from the lower-level generator. The following list shows some cases in which macro generators may call others; it may aid the programmer in interpreting such error messages:

Source Statement	*Lower-Level Macro-Instruction*
ARITH	Any function in the Macro Table
COMP	ARITH
EDMOV	MOVE
LOGIC	COMP, ZSIGN

Thus, if writing a LOGIC statement has resulted in an error message that is not listed or explained in the LOGIC macro-instruction description, the programmer should consult the descriptions of COMP and ZSIGN, and thereafter, since COMP in turn may have called ARITH, that of ARITH, etc.

Hardware Usage. Instructions generated from a macro-instruction may affect the following: (1) the contents of the three accumulators, (2) index words 93 and 94, in addition to such other index words as may be required (these will be assigned in the same fashion as are other symbolic index words), (3) latches as implied by the intent of the macro-instruction (Low, Equal, High for COMP, Accumulator Overflow, Field Overflow, and Sign Change for ARITH, etc.), (4) certain temporary storage areas reserved for working space, and (5) those fields or switches on which the macro-instruction is to operate, i.e., that are specifically named in the operand of the instruction in question. Wherever possible, these fields and switches will be treated non-destructively; thus logical variables in LOGIC, input fields in ARITH, the "from" fields in MOVE, etc. will preserve their contents during the execution of the generated instructions unless the contrary is indicated (e.g., the "to" field and the "from" field in SHIFT are identical). Such fields or switches as are intended to be affected (the result field in ARITH, switches in SETSW, etc.) will, of course, generally lose their previous contents, SNAP will leave the priority mask set to "allow," regardless of its former condition.

Since macro-instructions will often produce field overflows and sign changes, it is necessary to precede all programs employing macro-instructions with SMSC (Sense Mode Sign Change) and SMFV (Sense Mode Field Overflow) commands. If any segment of a program must be run in the halt mode, the latches should first be turned off with BFV (Branch Field Overflow) and BSC (Branch if Sign Change) commands, and the machine then placed in the halt mode. The sense mode must then be restored before macro-instructions are executed.

If the ARITH macro-instruction is used in a program, consideration must be given to the setting of the three Accumulator Overflow keys and the Exponent Overflow key (see page 132).

	Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–60) — Basic Autocoder	(61–75) — Autocoder
1.	01	ANYLABEL	MACRO	@.............................A VERY LON	G ALPHAMERIC LI
	02			TERAL@	
	03				
	04				
2.	05	ANYLABEL	MACRO	....	.......@NEW YORK
	06			CITY@	
	07				
	08				
3.	09	ANYLABEL	MACRO	....	......PARAMETER(
	10			XWORD+27)	
	11				
	12				
4.	13	ANYLABEL	MACRO	....	......PARAMETER
	14			(+125)	
	15				
	16				
5.	17	ANYLABEL	MACRO	....	PARAMETER(63+
	18			25)	
	19				
	20				
6.	21	ANYLABEL	MACRO	....	.....PARAMETER(
	22			29)	
	23				
	24				
7.	25	ANYLABEL	MACRO	.....................................PA	RAMETER(XWORD
				+12)	

OPEN

OPEN generates instructions to initialize input or output tape files for processing.

Source Program Format

The basic format for the OPEN statement is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	OPEN	FILE1, FILE2, FILE3, etc.
02			

ANYLABEL is any symbolic label; it may be omitted. The entry OPEN must be written exactly as shown. The operand must contain the name(s) of one or more tape files to be processed. Each name must be the same as the name which appears in the operand of the DTF entry which defines the file. As many tape files as desired may be named in the operand of an OPEN statement, provided the operand does not extend over more than four continuation cards. The names of the tape files must be separated by commas.

Processing Techniques

The first instruction generated by the processor as a result of an OPEN statement is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	BLX	IOCSIXG, IOC.IOPEN
02			

Following this, the processor will generate a branch constant containing the address of the first word of the File Specifications Table for each file named in the operand. A NOP will be generated after the last branch constant.

This calling sequence and the subroutine IOC.IOPEN (normally included as a result of a DIOCS statement) will perform the following operations:

1. Furnish details about the file to the File Scheduler routine.
2. Check on the availability of the file to the program.
3. Rewind the tape, if necessary.
4. Process the tape label, if any.
5. Mark the file as "active."

These operations will be performed automatically for subsequent reels of multi-reel files. In addition, end-of-reel operations (rewinding, writing of tape marks, and writing of trailer labels (if any)) will be instituted.

Error Messages

The following error messages will be produced during assembly under the conditions specified:

OPERAND BLANK

If the operand is blank, a NOP will be generated instead of the calling sequence.

PARAM*nn* NOT A FILE

If an operand parameter (the number is indicated by *nn*, above) is not defined by a DTF entry, a NOP will be generated at the point in the calling sequence where a branch constant would normally be included. Since the IOC.IOPEN subroutine would consider this NOP to be the end of the list of files to be initialized by OPEN, a manual correction must be made before the object program is run. If corrections are not made and the file is not the last one named in the operand of the OPEN statement, the object program will execute the next branch constant as a true Branch instruction, thus transferring control (in error) to the first word of the corresponding DTF.

GET generates instructions to obtain a record for processing.

Source Program Format

The basic formats for the GET statement are as follows:

Line	Label	Operation	OPERAND
01	ANYLABEL	GET	TAPEFILE
02	ANYLABEL	GET	TAPEFILE TO WORKAREA
03	ANYLABEL	GET	CARDFILE
04	ANYLABEL	GET	CARDFILE TO WORKAREA
05			

ANYLABEL is any symbolic label; it may be omitted. The entries GET and TO must be written exactly as shown. The first item in the operand must be the name of a tape file or unit record file. This name must be the same as the name which appears in the operand of the DTF or DUF entry which defines the file.

If the second or fourth format is used, the second operand item must be the word TO, preceded and followed by a single blank character. The third operand item must then be a name which appears as the label of a DRDW statement, the label of a DA header line, or the label of a DA subsequent entry.

Processing Techniques

The calling sequence generated by each GET statement, in conjunction with the File Schedulers and other subroutines of the Input/Output Control System, make it possible for the object program to obtain each input record one at a time, regardless of the form for the input. The record will be made available in the input area and, if specified, moved to a work area. When the third item in the operand is the label of a DRDW, the record will be moved to the area defined by the generated RDW. When the third item in the operand is the label of a DA header line, the record will be moved into the area defined by the DA. If the DA header line does not specify the generation of an RDW, the processor will generate (elsewhere) an RDW to be used by the GET statement. When the third item in the operand is the label of a DA subsequent entry, the input record will be moved into the area defined by that entry. In this case, the processor will always generate an RDW to be used by the GET statement.

For the second and fourth formats, the amount of the input record moved is completely dependent on the size of the area defined by the third item in the operand. No warning message will be issued if the size of the work area is not equal to the size of the input record. The programmer must define the work area to equal the amount of the input record that is to be moved.

Card Files

For card files, the GET statement using the third format causes the generation of the following instructions:

ANYLABEL	GET	CARDFILE
ANYLABEL	BLX	IOCSIXH, IOC.Dn

The GET statement using the fourth format causes the generation of the following instructions:

ANYLABEL	GET	CARDFILE TO WORKAREA
ANYLABEL	BLX	IOCSIXH, IOC.Cn
	B	WORKAREA

IOC.Cn and IOC.Dn are the labels of the two entry points to the unit record routine which is generated by the DUF entry that defines the file.

Tape Files

For tape files, the use of the first or second format will cause the address of the first word of the next record to be placed into the indexing portion of an index word specified in the File Specifications Table of the input file. Processing may then be done by using instructions referring to fields within the record as defined by a DA subsequent entry relative to 0000 plus this index word. This indexing will be assigned automatically by writing the name of the index word in the DA header line operand as described on page 35. If processing is to be done by using non-indexed instructions, the record may be moved to a work area by means of the MOVE macro-instruction. However, the second format above will initialize the index word *and* move the record to the work area. With either the first or second format, reading in of the next block of records from tape when all records in the input area have been processed is automatic.

Error and Warning Messages

The following error and warning messages will be produced during assembly under the conditions specified:

OPERAND BLANK

If the operand is blank, a NOP will be generated instead of the calling sequence.

OPERAND HAS TWO PARAMETERS

If there are two parameters in the operand, the calling sequence for the GET INPUTFILE form will be generated, followed by a NOP.

PARAM 01 NOT A FILE

If the first item in the operand is not the name of a tape or card file, a NOP will be generated instead of the calling sequence.

PARAM 01 NOT INPUT FILE

If a unit record file is named in the operand and if it is not an input file, this message will be produced. A NOP will be generated instead of the calling sequence.

PARAM 03 IS A FILE. PARAM 03 IGNORED

If the third item in the operand is the name of a tape or unit record file, the calling sequence for the GET INPUTFILE form will be generated, followed by a NOP.

PARAM 03 NOT DEFINED

If the third item in the operand is not defined by a DA header line, DA subsequent entry, or a DRDW statement, the calling sequence for the GET INPUTFILE form will be generated, followed by a NOP.

SRBFORM4 BLANK, ASSUMED 10

If the file named in the operand specifies Form 4 records, and if the subrecord blocking factor is not specified, a subrecord blocking factor of 10 will be assumed. The calling sequence, however, will be generated in the normal manner.

WARNING—PARAM 01 NOT INPUT FILE

If a tape file is named in the operand and if it is not an input file, this message will be produced. However, the calling sequence will be generated in the normal manner.

WARNING—PARAM 02 IS NOT -TO-

If there are three parameters in the operand, and if the second parameter is not the word "TO," this message will be produced. The calling sequence will be generated as if the second parameter *had* been "TO."

WARNING — RLIFORM3 BLANK

If the file named in the operand specifies Form 3 records, and if the record length indicator is not specified, this message will be produced. It will be assumed that the record length indicator is located in position 0 of word 0 of the record. In all other respects, the calling sequence will be generated in the normal manner.

PUT

PUT causes the generation of instructions that will provide the address of the next available word in the output area and, if desired, include a processed record in an output file by moving the record to the output area.

Source Program Format

The basic formats for the PUT statement are as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	PUT	OUTAPEFILE
02	ANYLABEL	PUT	TAPEFILE IN OUTAPEFILE
03	ANYLABEL	PUT	WORKAREA IN OUTAPEFILE
04	ANYLABEL	PUT	FIELDNAME IN OUTAPEFILE
05	ANYLABEL	PUT	CARDFILE IN OUTAPEFILE
06	ANYLABEL	PUT	OUTCARDFLE
07	ANYLABEL	PUT	TAPEFILE IN OUTCARDFLE
08	ANYLABEL	PUT	WORKAREA IN OUTCARDFLE
09	ANYLABEL	PUT	CARDFILE IN OUTCARDFLE
10			

ANYLABEL is any symbolic label; it may be omitted. The entries PUT and IN must be written exactly as shown. The operand may contain either one or three parameters, as follows:

1. If the operand contains one parameter it must be the name of an output tape or unit record file; it must appear as the operand of the DTF or DUF entry which defines the file.
2. If the operand contains three parameters, the third parameter must be the name of an output tape or unit record file; it must appear as the operand of the DTF or DUF entry which defines the file. The second parameter must be the word IN, preceded and followed by a single blank character. The first parameter may be defined by appearing as a name in any of the following:
   a. Operand of a DTF (the first format above).
   b. Operand of a DUF (the sixth format above).
   c. Label of a DA header line.
   d. Label of a DA subsequent entry.
   e. Label of a DC header line.
   f. Label of a DC subsequent entry.
   g. Label of a DLINE header line.
   h. Label of a DLINE subsequent entry.
   i. Label of a DRDW. (The area defined will be included in the output file.)

Processing Techniques

The calling sequences generated by each PUT statement, in conjunction with the File Schedulers and other subroutines of the Input/Output Control System, make it possible for the object program to cause the inclusion of each processed

record or field in the output file one at a time, regardless of the output blocking factor. If the name of the item to be included is defined by a declarative statement, and, if an RDW(s) is not specified for that item, the processor will generate (elsewhere) an RDW to be used by the PUT statement.

The use of the first or sixth format will cause the address of the next available word in the output area to be placed in the indexing portion of an index word specified in the File Specifications Table of the output file. Data may then be included in the output area by means of a later MOVE statement. (It may also be processed there until the next PUT occurs.) The other formats, however, not only cause the address of the next available word to be placed in the index word specified, but also cause automatic inclusion of the record in the output file and updating of the proper index words.

Writing of the output area will occur automatically when the area is full.

Error and Warning Messages

The following error and warning messages will be produced during compilation under the conditions specified:

OPERAND BLANK

If the operand is blank, a NOP will be generated instead of the calling sequence.

OPERAND HAS TWO PARAMETERS

If there are two parameters in the operand, a NOP will be generated instead of the calling sequence.

OUTPUT SRBFORM4 BLANK, ASSUMED 10

If the output tape file named in the operand specifices Form 4 records, and if the subrecord blocking factor is not specified, a subrecord blocking factor of 10 will be assumed. The calling sequence, however, will be generated in the normal manner.

PARAM 01 (03) NOT A FILE

If the parameter named as the output file is not a tape or unit record file, a NOP will be generated instead of the calling sequence.

PARAM 01 (03) NOT OUTPUT FILE

If the parameter named as the output file is a unit record file, and if it is not defined by its DUF as an output file, this message will be produced. A NOP will be generated instead of the calling sequence.

PARAM 01 UNDEFINED

If there are three parameters in the operand, and if the first parameter is not defined by one of the nine entries named under "Source Program Formats," this message will be produced. The calling sequence for the PUT OUTPUTFILE form will be generated, followed by a NOP.

WARNING — OUTPUT RLIFORM3 BLANK

If the output tape file named in the operand specifies Form 3 records, and if the record length indicator is not specified, this message will be produced. It will be assumed that the record length indicator is located in position 0 of word 0 of the record, and the calling sequence will otherwise be generated in the normal manner.

WARNING — PARAM 01 (03) NOT OUTPUT FILE

If the parameter named as the output file is a tape file, and if it is not defined by its DTF as an output file, this message will be produced. However, the calling sequence will be generated in the normal manner.

WARNING — PARAM 02 IS NOT -IN-

If there are three parameters in the operand, and if the second parameter is not the word "IN," this message will be produced. The calling sequence will be generated as if the second parameter had been "IN."

PUTX

PUTX causes the generation of instructions that will include a processed record in an output file by exchanging RDWs rather than by moving the record.

Source Program Format

The basic format for the PUTX statement is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	PUTX	INPUTFILE IN OUTPUTFILE
02			

ANYLABEL is any symbolic label; it may be omitted. The entries PUTX and IN must be written exactly as shown.

The first item in the operand must be the name of an input tape file. This name must appear in the operand of the DTF entry which defines the file.

The second item in the operand must be the word IN, preceded and followed by a single blank character.

The third item in the operand must be the name of an output tape file.
This name must appear in the operand of the DTF entry which defines the file.

Processing Techniques

The calling sequences generated by each PUTX statement, in conjunction with the File Schedulers and other subroutines of the Input/Output Control System, make it possible for the object program to cause the inclusion of each processed record in the output file, one at a time, regardless of the output blocking factor. Unlike the PUT statement, however, the PUTX statement causes this inclusion by the interchange of RDWs; the record itself is not moved. Thus, an RDW describing the record to be written is placed in the list of output RDWs and the RDW which was previously at that point in the list is placed back in the input list, replacing the original RDW.

The form of the records in the files places the following restrictions on the use of the PUTX macro-instructions:

1. Form 3 records *can not* be processed with a PUTX macro-instruction.
2. The combination of input record form and output record form must be one of the following:

Input File Record Form	*Output File Record Form*
1	1
1	2
2	1
2	2
4	4

3. The length of fixed length records or the maximum length of variable length records must be identical for both the input and output files.

4. For Form 4 records, the number of sections in input and output records must be the same and the maximum number of words in the corresponding sections of each input and output record must be identical.

There is no restriction on the blocking factor of the input and output files. The blocking factor, i.e., the number of records in one block, may be different for each of the files provided that the other restrictions listed above are observed.

After RDWs have been exchanged by a PUTX macro-instruction, the input record is no longer available for processing; the programmer must be certain that all processing requiring the input data is completed before issuing the command.

When PUTX is to be used, processing should be done in the input area using indexed instructions which refer to fields within the record as defined by a DA entry relative to 0000. If input data is moved to a work area for processing, do not use PUTX. If PUTX is used, the original input data rather than the results of processing will appear in the output file.

The automatic function of writing blocks of records on tape is the same for the PUTX macro-instruction as for the PUT macro-instruction.

Error and Warning Messages

The following error and warning messages will be produced during compilation under the conditions specified:

IMPROPER OPERAND

If the operand does not contain three parameters, a NOP will be generated instead of the calling sequence.

PARAM 01 (03) FILE FORM INVALID

If the files named in the operand do not conform to the restrictions regarding record form which are listed under "Processing Techniques," a NOP will be generated instead of the calling sequence.

PARAM 01 (03) NOT A FILE

If either the first or third item in the operand is not defined by a DTF, a NOP will be generated instead of the calling sequence.

PARAM 03 – SRBFORM4 BLANK, ASSUMED 10

If the output tape file named in the operand specifies Form 4 records, and if the subrecord blocking factor is not specified, a subrecord blocking factor of 10 will be generated. The calling sequence, however, will be generated in the normal manner.

RECLENGTHS UNEQUAL

If the record lengths specified for the tape files named in the operand are not equal, this message will be produced. The calling sequence will be generated in the normal manner, however.

SRBFORM4 UNEQUAL. OUTPUT SRB USED

If the tape files named in the operand specify Form 4 records, and if the subrecord blocking factors of the files are not equal, this message will be produced. The subrecord blocking factor of the output file will be used in the calling sequence.

CLOSE

CLOSE generates instructions to remove tape files from use.

Source Program Format

The basic format of the CLOSE statement is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	CLOSE	FILEA, FILEB, FILEC, etc.
02			

ANYLABEL is any symbolic label; it may be omitted. The entry CLOSE must be written exactly as shown. The operand must contain the name(s) of one or more tape files to be removed from processing use. Each name must be the same as the name which appears in the operand of the DTF entry which defines the file. As many tape files as desired may be named in the operand of a CLOSE statement, subject only to the restriction that the operand may not be extended over more than four continuation cards. The names of the tape files must be separated by commas.

Processing Techniques

The first instruction generated by the processor as a result of a CLOSE statement is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	BLX	IOCSIXG, IOC.ICLOSE
02			

Following this, the processor will generate a branch constant containing the address of the first word of the File Specifications Table for each file named in the operand. A NOP will be generated after the last branch.

This calling sequence and the subroutine IOC.ICLOSE (normally included as a result of a DIOCS entry) will perform the following for each output file:

1. Write out remaining records in the output area(s).
2. Write a tape mark.
3. Write end-of-file trailer labels (if desired).

In addition, the following will be performed for all files:

1. Rewind if necessary.
2. Mark the file as "inactive."

The CLOSE statement will normally be used to cause these operations to be performed for the *last* reel of each file.

Error Messages

The following error messages will be produced during assembly under the conditions specified:

OPERAND BLANK

If the operand is blank, a NOP will be generated instead of the calling sequence.

PARAM*nn* NOT A FILE

If an operand parameter (the number is indicated by *nn,* above) is not defined by a DTF entry, a NOP will be generated at the point in the calling sequence where a branch constant would normally be included. Since the IOC.ICLOSE subroutine would consider this NOP to be the end of the list of files to be removed by CLOSE, a manual correction must be made before the object program is run. If corrections are not made and the file is not the last one named in the operand of the CLOSE statement, the object program will execute the next branch constant as a true Branch instruction, thus transferring control (in error) to the first word of the corresponding DTF.

END

END generates instructions to remove tapes from use, type an end-of-job message, and then branch, halt, or permit SPOOL operations.

Source Program Format

The basic format for the END statement is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	END	BRANCHADDR
02			

ANYLABEL is any symbolic label; it may be omitted. The entry END must be written exactly as shown. The operand may be blank, or it may contain an actual or symbolic address.

Processing Techniques

The END statement will cause the generation of one of the following calling sequences:

```
ANYLABEL    END
ANYLABEL    BLX    IOCSIXG, IOC.IEND
            NOP    0
            or
ANYLABEL    END
ANYLABEL    BLX    IOCSIXG, IOC.IEND
            B      BRANCHADDR
```

These instructions and the IOC.IEND subroutine (normally included as a result of a DIOCS statement) will initiate the operations generally performed by the CLOSE macro-instruction for all files for which this has not yet been done, type an end-of-job message, and cause a branch to BRANCHADDR (if one is named).

If the branch address is omitted, the generated instructions will perform the necessary CLOSE operations; thereafter, the instructions will cause one of the following:

1. A halt if no SPOOL program is run in conjunction with the main program.
2. A program loop to be entered to permit SPOOL programs to continue if any are being run. Loading of another main program can then be initiated on signal from the SPOOOL routine.

ARITH—Arithmetic Operations

ARITH generates instructions to compute the value of an arithmetic expression and to store the result in any desired field.

Source Program Format

The basic formats for the ARITH statement in the source program are as follows:

Line 3–5	Label 6–15	Operation 16–20	OPERAND 21 25 30 35 40 45
01	ANYLABEL	ARITH	RESULT=EXPRESSION
02	ANYLABEL	ARITH	RESULT=EXPRESSION,OVERFLOWBR
03			

ANYLABEL is any symbolic label; it may be omitted. The entry ARITH and the equal sign must be written exactly as shown. The equal sign indicates that the RESULT field is to be set equal to the value of the arithmetic EXPRESSION. RESULT may be any symbolic name; the various allowable forms of the EXPRESSION are described under "Arithmetic Expressions," below. OVERFLOWBR is the symbolic label of the first instruction of a routine to which the program is to branch in the event of an overflow.

Arithmetic Expressions

Arithmetic expressions are formed from *numerical quantities* which may be in one of several *modes*. In addition, *arithmetic operators* or *functions* may operate on the quantities and *arithmetic punctuation* may establish an execution sequence.

Numerical Quantities

The numerical quantities that enter the computation may be of several kinds, namely symbolic fields (representing variables), literals, or defined constants. Fields referenced by symbolic names will generally be defined under a DA or DC header line, with their mode and (where applicable) format specified. Literals may be signed or unsigned; if unsigned, they will be interpreted as positive. Adcons will be treated as four-place integers in the automatic-decimal mode.

Arithmetic Operators

The ARITH statement interprets two types of arithmetic operators, unary and binary. A unary operator identifies or changes the sign of *one* numerical quantity; a binary operator indicates an operation to be performed upon *two* quantities to form another.

The following two *unary* operators are written preceding the quantity upon which they are to operate:

Operator	*Operation*
& or +	The ampersand or plus sign preceding a literal identifies it as positive. When either precedes another quantity, it does not affect its value.
–	A minus sign preceding a positive quantity changes its sign to minus; a minus sign preceding a negative quantity changes its sign to plus

The following five *binary* operators are written between the two quantities upon which they are to operate:

Operator	*Operation*
& or +	The ampersand or plus sign indicates that the two numbers are to be added.
—	The minus sign indicates that the second number is to be subtracted from the first number.
*	The asterisk indicates that the two numbers are to be multiplied.
/	The slash indicates that the first number is to be divided by the second number.
**	Double asterisks indicate exponentiation; i.e., the first number is to be raised to the power indicated by the second number.

The use of arithmetic operators preserves the mode of the operands.

Modes

Computation can be carried out in either automatic-decimal or floating-decimal mode. All numerical quantities appearing in EXPRESSION should be in the same mode. Two exceptions (4 and 5, below) are included in the following rules which apply to modes within EXPRESSION:

1. If A is a numerical quantity, then +A, −A, and ABS(A) are quantities of the same mode as A.
2. Enclosing a quantity(ies) in parentheses does not change the mode of the quantity(ies)
3. Adding, subtracting, multiplying, or dividing two quantities of the same mode will give a result in the same mode.
4. If A and B are numerical quantities, then A**B will be a permissible expression of the same mode as A, regardless of the mode of B. The permissible combinations of A and B and the resultant mode of A**B are indicated in the following table:

Mode of A	*Mode of B*	*Mode of A**B*
automatic-decimal	automatic-decimal	automatic-decimal
automatic-decimal	floating-decimal	automatic-decimal
floating-decimal	automatic-decimal	floating-decimal
floating-decimal	floating-decimal	floating-decimal

5. Arguments of functions may be in a different mode from that of the rest of EXPRESSION. (For details, see "Functions," below.)

The RESULT field may be in a different mode from that of EXPRESSION. In such a case, coding will be generated to edit the answer to the mode and format of the RESULT field.

The ARITH macro generator will not accept fields in the alphameric mode as operands. When an alphameric field, either literal or symbolic, is encountered, an error message will be issued and a NOP instruction generated.

Functions

During the computation of an arithmetic EXPRESSION, functions may be evaluated provided that appropriate subroutines are available. Such subroutines must be in the Program Library, even when they are specially provided by the programmer. (The function names used in the various examples of the ARITH macro-instruction are intended for illustration only. Their use should not imply that subroutines evaluating these functions are being furnished.)

The maximum number of arguments (independent variables) allowed for a function is equal to 69 minus the *total* number of parameters in the source statement. If a function is included in the source statement and this function with its arguments are the *only* parameters in the statement, then a maximum of 34 arguments would be allowed for the function. This is deduced since 34 arguments plus 1 function name equals 35 parameters and 69 minus 35 equals 34 arguments that will be accommodated. Under the same circumstances, if the function had 35 arguments, then 35 plus the 1 function name equals 36 parameters and 69 minus 36 equals only 33 arguments which can be accommodated. The last 2 arguments would be ignored and a message would be issued. If the source statement contains the maximum number of parameters permitted, i.e., 50, then a function included in the statement would be allowed only 19 arguments. Arguments may be considered to be either single variables (A, b, X, y) or expressions (a+b, 2*c/d−e, X**Y).

The arguments to be used by a function are enclosed by a set of parentheses and written following the function name. A comma is written following each argument but the last. An example of a function followed by a series of five arguments is as follows:

FUNCTION(A, b−2, X*Y, C**d, z)

ARITH will generate a call for the subroutine specified by the symbolic name FUNCTION, furnish the arguments to this subroutine and generate instructions for the computation to be continued, if necessary, after the function has been evaluated.

Function arguments constitute one of the exceptions to the rule that all fields within EXPRESSION must be in the same mode. For functions, this rule is modified as follows:

1. The argument(s) of any function must be in the proper mode in order for the specific subroutine called in to be able to operate upon it.
2. The function value must be in the proper mode so that the next computational step can be carried out.

The second condition states specifically that function values need not be in the mode of the EXPRESSION as a whole, since it is possible that the value obtained from one function will serve as an argument for another. When functions are nested in this manner, only the value of the outermost function must be in the mode of the EXPRESSION; the inner functions must produce values in a mode acceptable to the next function towards the outside. For example, if the expression A − ARCTAN (LOGE (X)) is to be evaluated, an Arctan function routine must be available whose output is in the same mode as A. A Log_e function must be available whose output is in a mode that the Arctan function can accept as an argument, and X must be in a mode that Log_e can accept as an argument. The result will be in the mode of A.

If a function has an expression as an argument, coding will be generated to compute this expression and to furnish the result to the function subroutine. Such computation, as well as arithmetic operations on function values prior to their being used as arguments of other functions, will preserve the original mode. For example, if 3 − SIN(3.1415 + ARCTAN(1 − X)) is to be computed, X must be in the automatic-decimal mode because 1 is in the automatic-decimal mode. The Arctan function must take arguments and yield function values in the same mode, and the same must be true of the Sine function. The result will then be automatic-decimal.

The absolute value function is a special function which is provided on the Compiler Systems Tape. This function is the only one whose instructions are generated in-line. The ABS function preceding a quantity indicates that the sign of the quantity is to be plus. Thus, a negative quantity is changed to a positive quantity; a positive quantity is unaffected. The quantity in question must be enclosed by parentheses.

Arithmetic Punctuation and Execution Sequence

Arithmetic expressions using more than one operator may be ambiguous unless the order in which the operations are to be carried out is indicated. For example, 12–2–3 yields 7 if the expression is interpreted to be (12–2)–3 and yields 13 if interpreted to be 12–(2–3). It is therefore necessary to establish an execution sequence of arithmetic operations by means of certain punctuation rules.

The ARITH statement will use parentheses in the customary way: expressions enclosed by parentheses are to be computed before they can be used as a component in the next operation. For nested parentheses (parentheses within parentheses) the same rule holds; the expression(s) enclosed by a greater number of parentheses will be computed before those enclosed by fewer parentheses.

To reduce the great number of parentheses which might be introduced by the explicit punctuation of all EXPRESSIONS, certain conventions are introduced. The following order of preference for operations is established:

1. Exponentiation
2. Unary Operators
3. Multiplication and Division
4. Addition and Subtraction

A sequence of operations will be carried out in this order unless parenthesization intervenes or directs otherwise.

If several operators of the same order are present, they will be executed from left to right. For example, A–B–C–D is treated as ((A–B)–C)–D; similarly, for multiplication and division, A/B/C*D is treated as ((A/B)/C)*D.

Repeated exponentiation is also executed from left to right unless punctuated otherwise. For example,

$2^{3^{2}}$ is interpreted to mean $(2^3)^2$ or 8^2, not $2^{(3^2)}$ or 2^9.

In other words, A**B**C is treated as (A**B)**C by the ARITH macro generator.

The only exception to this "left-to-right" rule applies to the addition and/or subtraction of automatic-decimal fields. The fields are taken in order of increasing number of decimal places rather than from left to right. This eliminates shifting the accumulator to the right which avoids loss of digits in this direction. Consequently, shifting will be only to the left; possible error is standardized to leftward overflow, which can be dealt with according to the procedures described under "Overflow Branch."

This execution order for operations is followed for expressions enclosed by the most parentheses, in order of decreasing parentheses, until the value of the entire EXPRESSION has been computed.

If these conventions are taken into account, a large number of parentheses may be omitted.

The generator will also accept statements in which unnecessary but correctly placed parentheses exist, but it is in the interest of storage economy to leave out parentheses wherever possible without introducing ambiguity. In no case may the number of parentheses (other than those used in address modification to enclose any one numerical field) exceed fifty. Since this limitation is identical to the number of permitted parameters, enough parentheses are allowed to punctuate any admissible expression. If the maximum parenthesis level is exceeded and the generator is unable to process the macro-instruction, an error message will be issued.

The following examples illustrate the use of the punctuation rules and the resultant sequence of execution:

EXPRESSION:	A—B*C
Interpretation:	A+((—B)*C)
Execution Sequence:	—B and C are multiplied; the result is added to A.
EXPRESSION:	(A—B)*C
Interpretation:	Same as EXPRESSION.
Execution Sequence:	B is subtracted from A; the result is multiplied by C.
EXPRESSION:	ABS(A)—B**C
Interpretation:	(ABS(A))—(B**C)
Execution Sequence:	B is raised to the power C; the absolute value of A is taken. Finally, the first result is subtracted from the second.
EXPRESSION:	A+B*C—D/3
Interpretation:	A+(B*C)—(D/3)
Execution Sequence:	B and C are multiplied; D is divided by 3; the necessary addition and subtraction operations are performed.
EXPRESSION:	A*1.2—SIN(X+3)/3+Y**4
Interpretation:	(A*1.2)—((SIN(X+3))/3)+(Y**4)
Execution Sequence:	X and 3 are added and the sine of the sum taken. Y is raised to the 4th power; A is multiplied by 1.2; the sine is divided by 3. Finally, the necessary addition and subtraction of the intermediate results are performed.

Processing Techniques

Limitations on Length

The number of permissible parameters is fifty. Parameters are considered to be the following:

1. Numerical fields (including function arguments).
2. Names of functions, including ABS.
3. Overflow branch, if specified.

Arithmetic operators and punctuation are *not* counted as parameters. An attempt to write more than the permited number of parameters in the operand of an ARITH statement will be intercepted by the processor and no coding will be generated.

Spacing and Punctuation

No blanks should appear in the operand of an ARITH macro-instruction. Only one equal sign may appear, positioned as shown under "Source Program Format."

Commas must separate function arguments; one additional comma is required to separate EXPRESSION from OVERFLOWBR if the second format is chosen. The appearance of illegal characters in the operand will cause an error message to be issued.

Address Modification

Symbolic addresses may be modified by indexing and address adjustment.

Overflow Branch

Since the conditions under which accumulator overflow could take place during a computation are extremely varied, no general routine for dealing with overflow has been provided. Instead, opportunity is given to the programmer to supply his own correction routine, which can be suited to his particular ARITH statement. If an overflow branch is specified, the generated instructions will contain a BLX instruction to cause a transfer to the overflow routine under the following conditions:

1. *Automatic-Decimal Computations:* Overflow resulting from addition, subtraction, exponentiation, store, and add-to-storage operations. (Transfer will *not* be caused by multiplication and division overflow. However, warning messages will be issued.)
2. *Floating-Decimal Computations:* Exceeding the maximum value for floating-decimal numbers as a result of *any* operation. This transfer will be made whether the object program machine has floating-decimal hardware or not. In the first case, the floating-decimal overflow indicator is tested; in the second, this test is simulated along with the floating-decimal arithmetic procedures.

Following the overflow routine, the program will return to the instruction in the object program following the BLX, provided that the last instruction in the overflow routine is an unconditional Branch to location 0000 + X94.

If no overflow branch is indicated, warning messages will be issued during automatic-decimal arithmetic assemblies pointing out the possibility of overflow.

Since the possibility of overflow in floating-decimal computations cannot be detected on the basis of format alone, warning messages will never be issued during floating-decimal arithmetic assemblies.

If OVERFLOWBR is indicated in the operand of the ARITH statement but, because of the input formats involved, overflow cannot possibly occur, then no overflow branch will be generated. This is illustrated in example 4 under "Examples."

Mode Size for Automatic-Decimal Computations

In handling automatic-decimal numbers, the processor establishes a mode size which is either ten or twenty digits in length and is based on the input formats of all the fields in the ARITH source statement. Then, based on the mode size which has already been determined and on the input formats, a "computation mask" or standard format is established. The mask indicates the decimal point placement and the maximum number of digits which may appear to the left and to the right of the decimal point. No intermediate results will be permitted to exceed this format.

Symbols have been established for this discussion and are defined as follows:

Symbol	*Definition*
MS	The mode size for automatic-decimal computations.
MDL	The maximum number of digits to the left of the decimal point (i.e., integer digits) as specified by the input formats in the ARITH source statement (both EXPRESSION and RESULT fields).
MDR	The maximum number of digits to the right of the decimal point (i.e., decimal digits) as specified by the input formats in the ARITH source statement (both EXPRESSION and RESULT fields).

Symbol	*Definition*
DL	The number of digits to the left of the decimal point (integer digits) in a mask.
DR	The number of digits to the right of the decimal point (decimal digits) in a mask.

The *mode size* for automatic-decimal computation is established as follows. If MDL exceeds 20, i.e., if any of the input fields has more than twenty integer digits, an error condition results; error message N 19 is issued and no coding other than a NOP will be generated.

If MDL is less than or equal to 20, the mode size depends on the sum of MDL and MDR, as follows:

MDL+MDR	*MS*
$\leq$10	10
$>$10	20

Once mode size has been established, the *mask* is defined. The procedure followed again depends on the value of MDL+MDR:

1. If MDL+MDR is less than or equal to 20, DR will be set equal to MDR, and DL will be set equal to MS—DR.

2. If MDL+MDR exceeds 20, DL will be set equal to MDL; DR will be set equal to MS—DL. Enough decimal digits will be truncated (without rounding) to reduce the overall length to twenty digits. A warning message (W 18) will include the computation mask in four-digit form, the first two representing the number of integers, the last two the number of decimals.

In short, if the total number of digits in all the input fields, with their decimal points aligned, is less than the computed mode size, the extra capacity is applied to the integer side. If the total exceeds 20, high-order digits are protected and decimal places truncated.

The following examples illustrate the methods of establishing computation masks:

1. Ten-digit mask with extra integer capacity.

Input Formats		
	MDL = 6	DR=MDR=3
5.2	MDR = 3	DL=MS—DR=10—3=7
6.1	Sum = 9	
3.3	MS = 10	*Mask: 7.3*

2. Ten-digit mask without extra capacity.

Input Formats		
	MDL = 8	DR=MDR=2
7.1	MDR = 2	DL=MS—DR=10—2=8
8.1	Sum = 10	
3.2	MS = 10	*Mask: 8.2*

3. Twenty-digit mask with extra integer capacity; input fields do not exceed ten digits.

Input Formats		
	MDL = 8	DR=MDR=7
2.3	MDR = 7	DL=MS—DR=20—7=13
8.2	Sum = 15	
1.7	MS = 20	*Mask: 13.7*

4. Twenty-digit mask with extra integer capacity; input fields exceed ten digits.

Input Formats		
	MDL = 9	DR=MDR=8
9.2	MDR = 8	DL=MS—DR=20—8=12
7.8	Sum = 17	
3.1	MS = 20	*Mask: 12.8*

5. Twenty-digit mask without extra capacity; input fields exceed ten digits. (This case could also occur with input fields not exceeding ten digits if two of them had formats of 10.0 and 0.10, respectively; the mask would then be 10.10.)

Input Formats		
	MDL = 4	DR=MDR=16
3.16	MDR = 16	DL=MS—DR=20—16=4
4.9	Sum = 20	
1.4	MS = 20	*Mask: 4.16*

6. Twenty-digit mask with decimal digits truncated.

Input Formats		
	MDL = 13	DL=MDL=13
13.2	MDR = 9	DR=MS—DL=20—13=7
11.8	Sum = 22	
4.9	MS = 20	*Mask: 13.7*

Two decimal places will be lost. Warning message W 18 will give the computation mask in the form 1307.

During computation, all intermediate results will be confined to the mask. Excess decimal digits developed will be truncated without rounding or warning. Excess integer digits may also be lost; if this becomes possible, the consequences will depend on the type of operation that caused the difficulty.

Addition or Subtraction. If an overflow branch has been specified, coding will be generated to transfer the object program to this branch if necessary. If no overflow branch is indicated, warning message W 21 or W 22 will be issued during assembly. Warning messages are issued if overflow or digit loss is *possible,* as determined on the basis of field *format* alone; the transfer to the overflow branch takes place only when these conditions become *actual* due to the specific object-time contents of the fields.

Multiplication or Division. If integer digits may possibly be lost during multiplication and division operations, warning message W 20 will be issued during assembly. Overflow resulting from multiplication or division will not cause a transfer to the overflow branch.

If the divisor field in a division operation is defined as having integer digits, the high-order digit is significant. In other words, if the automatic-decimal format of a divisor is 2.4, the generator will proceed on the assumption that the contents of the field at object program time will be *at least* 10.0000. If this condition is not satisfied, intermediate results may exceed the computation mask without a warning at assembly time.

If the divisor has an automatic-decimal format of the type 0.n, it will merely be assumed that the nth decimal digit contains at least a 1; this assumption is, of course, non-restrictive, since zero divisors lead to special procedures as described under "Zero Divisors."

When a warning message is issued, the programmer should check whether his intermediate results can exceed the mask on the left; this will depend on his

actual data as well as the defined field formats and the specified arithmetic operations. If overflow can occur, an input field can be redefined so as to (1) change a ten-digit computation to twenty-digit mode size, or (2) increase the number of integer digits at the expense of decimal digits. To avoid recurrence of the same problem after redefinition, it is generally advisable to modify the defined format of the result field where the program objectives permit.

Exponentiation. The exponentiation of all numbers (automatic-decimal or floating-decimal bases having either automatic-decimal or floating-decimal exponents) is carried out by means of floating-decimal routines. Automatic-decimal bases and exponents are converted to floating-decimal numbers and exponentiation is carried out by means of a subroutine. The result will be converted to an automatic-decimal number, if required. Three exceptions to this process are as follows:

1. If the base is an automatic-decimal integer and the exponent is an automatic-decimal integer less than ten digits in length, exponentiation is carried out by means of a subroutine which generates the required multiplication instructions.
2. If the base is an automatic-decimal integer and the exponent is a literal 2 or 3, Multiply instructions are generated in-line.
3. If the base is a floating-decimal number and the exponent is a literal 2 or 3, Floating Multiply instructions are generated in-line.

If the result of the exponentiation should exceed the computation mask format on the left, transfer will be made to an overflow branch when one is specified. Otherwise, the overflow latch for Accumulator 1 will be set ON. No warning message will be issued at assembly time (except for cases 1 and 2, above), since the size of the result cannot be predicted on the basis of floating-decimal field formats.

Zero Divisors

Before a division is executed, the divisor is tested for zero. If the divisor is zero, the overflow latch for Accumulator 3 is set ON. Subsequent procedures depend upon the mode of computation, as follows:

1. *Automatic-Decimal Computations:* The accumulator(s) containing the quotient will be filled with 9s and given the proper sign determined by the signs of the dividend and divisor. Computation will then continue as usual.
2. *Floating-Decimal Computations:* The division is ignored; i.e., the dividend is used as quotient. Computation will then continue as usual.

No other operation initiated by the ARITH macro-instruction alters the overflow latch for Accumulator 3, with the possible exception of function subroutines, whose individual specifications may be consulted. This latch, therefore, provides a certain test as to whether division by zero has been attempted. If such a test is desired, the latch must be set OFF before the ARITH macro-instruction is executed, since no automatic provision is made for this by the generator.

Final Storage

If the RESULT field is in a mode different from that of the computation, editing will be necessary before final storage. The following rules apply:

1. In editing from automatic-decimal to floating-decimal mode, the result will appear in normalized form. Only the first eight significant digits will be converted; further digits to the right will be truncated without rounding.
2. In editing from floating-decimal to automatic-decimal mode, the following three cases are distinguished:

a. If, after conversion, the first significant digit falls to the left of of the high-order digit of the RESULT field, an overflow condition exists. No warning message can be issued at assembly time since this condition cannot be predicted on the basis of floating-decimal field format alone. At object time, a message "SHIFT OUT OF RANGE, FLT TO DECI" will be typed out. Transfer will be made to the overflow branch if one is specified; such digits as can be accommodated in their proper places will be stored. If no overflow branch is specified, the overflow latch of Accumulator 1 will be set ON.
b. The "normal" case exists if, after conversion, the first significant digit falls into one of the digits of the RESULT field. Excess decimal digits developed beyond the capacity of the RESULT field will be truncated after rounding.
c. If, after conversion, the first significant digit falls to the right of the low-order digit of the RESULT field, then the decimal value of the answer is too small to register in the established format. At object time, a message "SHIFT OUT OF RANGE, FLT TO DECI" will be typed out. The RESULT field will be set to zero.

If the result of an automatic-decimal computation is to be stored in an automatic-decimal RESULT field of smaller format, the following procedures are followed:

1. Excess decimal digits are truncated *after* rounding; in other words, if the digit immediately to the right of the point of truncation contains a value of 5 or more, the next digit to the left is increased by 1.
2. Excess integer digits are lost and an overflow condition results. If an overflow branch is specified, transfer will be made to the overflow routine at object program time. If no overflow branch is indicated, warning message W 23 is issued during assembly.

Setting Overflow Lights

The overflow lights for Accumulators 1, 2, and 3 should be set as follows for both automatic-decimal and floating-decimal computations. The exponent overflow light should be set as indicated for floating-decimal computations if floating hardware is used.

Accumulator 1. If the accumulator 1 overflow light is *not* ON and Accumulator 1 overflows, the machine will stop.

If the accumulator 1 overflow light is ON, one of the following provisions should be made:

1. An overflow branch in the operand of the ARITH macro-instruction.
2. A BV1 (Branch if Overflow in Accumulator 1) instruction following each ARITH without an overflow branch.

If neither of the above provisions is made and the accumulator overflows, the condition will be carried and an error will be introduced.

Accumulator 2. The accumulator 2 overflow light must be ON during object program time.

Accumulator 3. The accumulator 3 overflow light must be ON if the detection of an attempt to divide by zero is desired.

Exponent. If the exponent overflow light is *not* ON and the exponent overflows, the machine will stop.

If the exponent overflow light is ON, one of the following provisions should be made:

1. An overflow branch in the operand of the ARITH macro-instruction.
2. A FBV (Floating Branch Overflow) instruction following each ARITH without an overflow branch.

If neither of the above provisions is made and the exponent overflows, the condition will be carried and an error will be introduced.

Error and Warning Messages

Under the conditions specified, the ARITH macro generator will issue the following error and warning messages during assembly. Unlike the other macro generators, ARITH does not give the text of the message. Only the message code letter and number, possibly supplemented by a parameter number or computation-mask format, are given. The programmer must refer to the list below for text and interpretation. The code letters are to be interpreted as follows:

Code	*Interpretation*
N	An error condition exists that makes further coding impossible; a NOP has been generated.
W	A warning that either an unusual condition or the possibility of error exists; generation continues.
X	An error condition exists; generation will continue on the special assumptions stated in the message.

N 01 NO OPERAND

No numerical field has been specified upon which an operation is to be performed.

X 02 NO EQUAL SIGN — WILL NOT STORE RESULT

The RESULT field and the equal sign have both been omitted. The generator has produced instructions to compute the value of the arithmetic EXPRESSION, but not to store the result.

X 03 NO RESULT FIELD — CANNOT STORE RESULT

The operand portion of the macro-instruction begins with the equal sign. The generator has produced instructions to compute the value of the arithmetic EXPRESSION, but not to store the result.

N 04 ALPHA FIELD UNACCEPTABLE. PARAMETER *xx*

The parameter number of the alphameric field has been included in the message.

X 07 INCOMPLETE — WILL PROCESS TO PARAMETER *xx*

The text of the input statement appears to be broken off; e.g., it ends with a left parenthesis. This condition may also occur if one blank precedes an entry that should be processed on the same card. The number of the last parameter processed has been included in the message.

X 08 TEXT ENDS WITH OPERATOR — WILL IGNORE

The last operator does not have an operand on its right and has been ignored.

X 09 CONSECUTIVE OPERATORS — WILL ACCEPT FIRST ONLY BEFORE PARAMETER *xx*

Two successive arithmetic operators have been detected by the scan, pre-

ceding the parameter whose number has been included in the message. The second of these operators has been ignored.

X 10 NO PUNCTUATION — WILL IGNORE PARAMETER *xx*

Two consecutive numerical fields have been detected, without an intervening operator. The second field, whose parameter number has been included in the message, has been ignored.

X 11 ILLEGAL CHARACTER — WILL BE IGNORED. BEFORE PARAMETER *xx*

The scan has detected a character that is not one of the allowable punctuation marks or arithmetic operators. The number of the next following parameter has been included in the message to aid in locating the faulty entry.

X12 EXCESS RIGHT PARENTHESES — WILL IGNORE

An attempt has been made to close more parentheses than had been opened. All right parentheses unmatched by left parentheses have been ignored.

X 13 END OF SCAN — PARENTHESES DO NOT MATCH — WILL SUPPLY

Some parentheses have been left open. Generation of instructions have proceeded on the assumption that they are all to be closed at the right end of the EXPRESSION.

X 16 FUNCTION WITHOUT ARGUMENT — PROCESSING STOPS BEFORE PARAMETER *xx*

A function symbol, whose parameter number has been included in the message, has not been followed by any argument. Neither the function nor any subsequent entries have been processed.

X 17 FUNCTION HAS TOO MANY ARGUMENTS — WILL IGNORE EXCESS. PARAMETER *xx*

An attempt has been made to write a function, whose parameter number has been included in the message, with more than the maximum number of arguments which can be handled by the statement. Only the arguments which can be accommodated have been passed on to the function subroutine.

W 18 DECIMAL DIGITS TRUNCATED. COMPUTATION MASK IS *IIDD*

In order to accommodate sufficient integer digits in computing intermediate results, some of the decimal places of the input fields must be truncated. The message includes the four-digit computation mask; the first two digits indicate the maximum number of integers, the last two digits the maximum number of decimal digits. If an input field has any decimal digits in excess of this maximum, the digits will be truncated without rounding.

N 19 AUTO-DECIMAL FIELD HAS MORE THAN 20 INTEGERS

Since the maximum mode size for automatic-decimal computation is twenty digits, input fields having more than twenty integers would lead to meaningless results. A NOP has been generated.

W 20 INTEGER DIGITS MAY EXCEED COMPUTATION MASK *IIDD*

An automatic-decimal computation may develop more integer digits than are provided for in the maximum format for intermediate results. This format has been included in the message in four-digit form, the first two digits representing the maximum number of integer digits that can be accommodated. (For details, remedies, etc., see "Mode Size for Automatic-Decimal Computations.")

W 21 SHIFT LEFT MAY LOSE HIGH-ORDER DIGIT(S)

Shifting left in the accumulator(s) to accommodate automatic-decimal num-

bers with more decimal places for addition or subtraction may have caused digit loss on the left. This message has been issued because no overflow branch has been specified.

W 22 ACCUMULATOR OVERFLOW POSSIBLE

An operation of addition or subtraction has been performed in the automatic-decimal mode that may have led to accumulator overflow. This message has been issued because no overflow branch has been specified.

W 23 OVERFLOW POSSIBLE IN RESULT FIELD

The number of integer digits that may have been developed in evaluating the arithmetic EXPRESSION exceeds the number available in the RESULT field. This message has been issued because no overflow branch has been specified. (Excess decimal digits will have been rounded off without warning.)

N 24 FIELD SIZE EQUAL TO 0

The parameter record has been incorrectly constructed by a higher level macro-instruction. A NOP has been generated.

X 25 MISSING OPERATOR AFTER FLOATING POINT LITERAL

Operand following that literal has been ignored.

X 26 UNUSUAL PARAMETER

The parameter was not usable to ARITH in its original form. The generator has produced instructions to treat the parameter as a 10-digit integer.

N 27 EQUAL SIGN IN MIDDLE OF EXPRESSION

Only one equal sign is allowed in a statement. A NOP has been generated.

W 28 INVALID REFERENCE TO A CODE SUBSEQUENT ENTRY

The RESULT field is a symbolic label of a CODE subsequent entry. Coding has been generated to store the result in the indicated literal.

Examples

The following are examples of acceptable coding for the ARITH macro-instruction. For each, the associated source-program entries are given, followed by the ARITH statement, coding generated in-line and (where applicable) coding generated out-of-line.

```
PAGE AA     PROGRAM                                  7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL       OP     OPERAND                                    CDNO FD  LOC  INSTRUCTION      REF

01 0054   *                  ARITH EXAMPLE 1
02 0055               DA     1                                                        +0003250327
03 0056   INPUT              12,19A5.3                                       29  0326                    0326
04 0057   DELTA              22,29A4.4                                       29  0327                    0327
05 0058   *
06 0059   ANYLABEL    ARITH  INPUT=DELTA
07      X ANYLABEL    ZA2    DELTA(0,7)                                 00001    0328 +2300290327
08      X             SRR2   1                                                   0329 +5000002101
09      X             ST2    INPUT(0,7)                                          0330 +2200290326
10 0060   *
```

```
PAGE AA     PROGRAM                                  7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL       OP     OPERAND                                    CDNO FD  LOC  INSTRUCTION      REF

01 0064   *                  ARITH EXAMPLE 2
02 0065               DA     1                                                        +0003250325
03 0066   X                  1,3A3.0                                         13  0325                    0325
04 0067   *
05 0068   ANYLABEL    ARITH  X=X+1                                                                         M
06      X ANYLABEL    ZA2    X(0,2)                                     00001    0326 +2300130325
07      X             A2     +1                                                  0327 +2400000329
08      X             ST2    X(0,2)                                              0328 +2200130325
09 0069   *
10 0070   *        THE FOLLOWING IS GENERATED OUT OF LINE
11 0071   *
                      LITERALS
12      X                    +1                                          00  0329 +1                       0329
```

```
ERROR MESSAGE LIST

PG/LN    MESSAGE

AA 05 ARITH W 20 0300
```

ARITH Example 2

Warning message W 20 has been produced because an automatic-decimal computation might develop more integer digits than can be accommodated by the mask 3.0, which has been included in the message in the form 0300.

```
PAGE AA    PROGRAM                                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL       OP    OPERAND                                                  CDNO FD  LOC  INSTRUCTION      REF

01 1469   *                 ARITH EXAMPLE 3
02 1471               DA    1                                                                      +0003250330
03 1472   A                 11,18A4.4                                                     18  0326                  0326
04 1473   B                 23,29A4.3                                                     39  0327                  0327
05 1474   C                 30,39A4.6                                                     09  0328                  0328
06 1475   D                 40,48A3.6                                                     08  0329                  0329
07 1476   X                 50,59A4.6                                                     09  0330                  0330
08 14761  OVERFLOWBR NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE    00001      0331 -0100090000
09 1477   *
10 1478   ANYLABEL    ARITH X=A-B+ABS(C-D),OVERFLOWBR
11      X ANYLABEL    ZS2   D(0,3)                                                            0332 -2300030329
12      X             SL2   6                                                                 0333 +5000002206
13      X             A2    C(0,9)                                                            0334 +2400090328
14      X             MSP   9992                                                              0335 -0300919992
15      X             ST2   COMAREA.A(0,9)+1                                         00002    0336 +2200090348
16      X             ZS2   B(0,3)                                                            0337 -2300360327
17      X             SL2   4                                                                 0338 +5000002204
18      X             A2    A(0,7)                                                            0339 +2400180326
19      X             SL2   2                                                                 0340 +5000002202
20      X             A2    COMAREA.A(0,9)+1                                         00003    0341 +2400090348
21      X             BV2   M.1                                                               0342 +2100090345
22      X             BV3   M.1                                                               0343 +3100090345
23      X             B     *+2                                                               0344 +0100090346
24      X M.1         BLX   93,OVERFLOWBR                                                     0345 +0200930331
25      X             ST2   X(0,9)                                                   00004    0346 +2200090330
26 1479   *
27 1480   *        THE FOLLOWING IS GENERATED OUT OF LINE
28 1481   *
29      X COMAREA.A   DA                                                                           +0003470348
```

```
PAGE AA     PROGRAM                                        7070 COMPILER SYSTEM VERSION UMY22, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL        OP     OPERAND                                                    CDNO FD  LOC  INSTRUCTION     REF

01 0075   *                   ARITH EXAMPLE 4.
02 0076   OVERFLOWBR NOP            REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE     00001     0325 -0100090000
03 0077                DA     1                                                                      +0003260330
04 0078   A                   11,19A5.4                                                        19  0327                  0327
05 0079   B                   23,29A4.3                                                        39  0328                  0328
06 0080   C                   30,39A4.6                                                        09  0329                  0329
07 0081   X                   42,49A6.2                                                        29  0330                  0330
08 0082   *
09 0083   ANYLABEL     ARITH  X=A+B-C,OVERFLOWBR
10      X ANYLABEL     ZA1    +0                                                         00002     0331 +1300000366
11      X              ZA2    B(0,6)                                                               0332 +2300390328
12      X              BLX    94,LINK.A                                                            0333 +0200940343
13      X              SL     1                                                                    0334 -5000000201
14      X              A2     A(0,8)                                                               0335 +2400190327
15      X              BLX    94,LINK.A                                                  00003     0336 +0200940343
16      X              SL     2                                                                    0337 -5000000202
17      X              S2     C(0,9)                                                               0338 -2400090329
18      X              BLX    94,LINK.A                                                            0339 +0200940343
19      X              SRR    4                                                                    0340 -5000000104
20      X              ST2    X(0,7)                                                     00004     0341 +2200290330
21 0084   *
22 0085   *       THE FOLLOWING IS GENERATED OUT OF LINE
23 0086   *
24      X              B      LINK8.A+1                                                            0342 +0100090365
25      X LINK.A       BV2    LINK1.A IS THERE A CARRY                                             0343 +2100090351
26      X LINK3.A      BZ2    LINK4.A                                                              0344 +2000090348
                       BZ1    LINK5.A                                                              0345 +1000090349
                                                                                                   0346 -1000090356

                                                                                                   0362 +2400090365
45      X              A2     +1 CHANGE 9992                                                       0363 +2400110366
46      X LINK8.A      B      0+X94                                                                0364 +0194090000
                       LITERALS
47      X                     +9999999999                                                  09  0365 +9999999999     0365
48      X                     +0                                                         00009 00  0366 +0              0366

01      X                     +1                                                               11  0366 + 1             0366
```

ARITH Example 4

Since MDL+MDR is greater than 10 in the fields defined in this example, the computation will be in twenty-digit mode, with a mask of 14. 6.

```
PAGE AA    PROGRAM                                     7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA
LN CDREF  LABEL       OP    OPERAND                                        CDNO FD  LOC  INSTRUCTION    REF

01 1496   *                 ARITH EXAMPLE 5
02 1497               DA    1                                                            +0003250328
03 1498   FA                00,09F                                               09  0325                0325
04 1499   FB                10,19F                                               09  0326                0326
05 1500   FC                20,29F                                               09  0327                0327
06 1501   FRESULT           30,39F                                               09  0328                0328
07 1502   *
08 1503   ANYLABEL    ARITH FRESULT=SQRT(FA+FB)+FC**2                                                       M
09      X ANYLABEL    ZA1   FA(0,9)                                         00001      0329 +1300090325
10      X             A2    FB(0,3)                                                    0330 +2400030326
11      X             ST1   COMAREA.A(0,9)+1                                           0331 +1200090397
12      X             ZA1   COMAREA.A(0,9)+1                                           0332 +1300090397
13      X             BLX   94,SQRT.A                                                  0333 +0200940354
14      X             ST1   COMAREA.A(0,9)+1                                00002      0334 +1200090397
15      X             ZA2   FC(0,3)                                                    0335 +2300030327
16      X             ZA3   9992                                                       0336 +3300099992
17      X             M     9993                                                       0337 +5300099993
18      X             SLC   MACREG.1                                                   0338 -5000010300
19      X             ZA3   MACREG.1(4,5)                                   00003      0339 +3300450001
20      X             C3    +0000000016                                                0340 +3500090399
21      X             BL    *+4                                                        0341 +4000090345
22      X             XS    MACREG.1,0                                                 0342 -4700010000
23      X             SRR   0+MACREG.1                                                 0343 -5001000100
24      X             B     *+5                                             00004      0344 +0100090349
25      X             ZA1   +9999999999                                                0345 +1300090401
26      X             ZA2   +9999999999                                                0346 +2300090401
27      X             A1    +5000000000                                                0347 +1400090400
28      X             A1    +5000000000                                                0348 +1400090400
29      X             ST2   COMAREA.A(0,9)+2                                00005      0349 +2200090398
30      X             ZA1   COMAREA.A(0,9)+1                                           0350 +1300090397
31      X             A2    COMAREA.A(0,9)+2                                           0351 +2400090398
32      X             ST1   FRESULT(0,9)                                               0352 +1200090328
33 1504   *
34 15041  *           THE FOLLOWING IS GENERATED OUT OF LINE
35 15042  *
36      X             B     M.10+1                                                     0353 +0100090396
37      X SQRT.A      BZ1   0+X94                                           00006      0354 +1094090000
38      X             BM1   M.4                                                        0355 -1000090383
```

```
25      X M.7               +5000000000                                          09  0387
26      X M.8               +6250000001                                          09  0388 +6250000001  0388
27      X M.9               +0101501025                                          09  0389 +0101501025  0389
28      X                   +0201930800                                          09  0390 +0201930800  0390
29      X                   +0702750550                                     00013 09  0391 +0702750550  0391
30      X                   +1704350350                                          09  0392 +1704350350  0392
31      X                   +2906180250                                          09  0393 +2906180250  0393
32      X                   +5907600200                                          09  0394 +5907600200  0394
33      X M.10              +9911230138                                          09  0395 +9911230138  0395
34      X COMAREA.A   DA                                                                  +0003960398
              LITERALS
35      X                   +0000000016                                     00014 09  0399 +0000000016  0399
36      X                   +5000000000                                          09  0400 +5000000000  0400
37      X                   +9999999999                                          09  0401 +9999999999  0401
```

```
ERROR MESSAGE LIST

PG/LN    MESSAGE

AA 08 ARITH W 20 0400
```

```
PAGE AA     PROGRAM                                  7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL       OP    OPERAND                                            CDNO FD   LOC  INSTRUCTION       REF

01 0088   *                 ARITH EXAMPLE 6.
02 0089               DA    1                                                                  +0003250328
03 0090   PRESSURE          00,07A4.4                                                07   0325                   0325
04 0091   VOLUME            10,15A2.4                                                05   0326                   0326
05 0092   CENTIGRADE        20,25A2.4                                                05   0327                   0327
06 0093   CONSTANT          30,39A6.4                                                09   0328                   0328
07 0094   *
08 0095   ANYLABEL    ARITH CONSTANT=PRESSURE*VOLUME/CENTIGRADE
09      X ANYLABEL    ZA3   PRESSURE(0,7)                                      00001      0329 +3300070325
10      X             M     VOLUME(0,5)                                                   0330 +5300050326
11      X             SR    4                                                             0331 -5000000004
12      X             ZA3   CENTIGRADE(0,5)                                               0332 +3300050327
13      X             BLX   93,DIV1.A                                                     0333 +0200930338
14      X             XA    MACREG.1,0+5                                       00002      0334 +4700010005
15      X             B     DIV2.A                                                        0335 +0100090346
16      X             ST2   CONSTANT(0,9)                                                 0336 +2200090328
17 0096   *
18 0097   *        THE FOLLOWING IS GENERATED OUT OF LINE
19 0098   *
20      X             B     DIV3.A+6 TEMP BRANCH FOR GENOS                                0337 +0100090370
21      X DIV1.A      BZ3   DIV3.A                                                        0338 +3000090364
22      X             ZA1   +0                                                 00003      0339 +1300000372
23      X             SLC   MACREG.1                                                      0340 -5000010300
24      X             SR    1                                                             0341 -5000000001
25      X             SLC3  MACREG.2                                                      0342 +5000023300
26      X             XS    MACREG.1,10+MACREG.2                                          0343 -4702010010
27      X             D     9993                                               00004      0344 -5300099993
28      X             B     0+X93                                                         0345 +0193090000
29      X DIV2.A      BXM   MACREG.1,DIV4.A                                               0346 -4400010355
30      X             ZA1   MACREG.1(4,5)                                                 0347 +1300450001
31      X             C1    +11                                                           0348 +1500120372
32      X             BL    *+3                                                00005      0349 +4000090352
33      X             SR2   10                                                            0350 +5000002010
34      X             B     2+X93                                                         0351 +0193090002
35      X             STD1  *(8,9)+1                                                      0352 -1200890353
36      X             SRR2  10                                                            0353 +5000002110
37      X             B     2+X93                                              00006      0354 +0193090002
38      X DIV4.A      SLC2  MACREG.2                                                      0355 +5000022300
39      X             MSP   MACREG.1                                                      0356 -0300910001
                      XS    MACREG.2,0+MACREG.1                                           0357 -4701020000

01      X             SL    0                                                             0366 -5000000200
02      X             ZA2   +9999999999                                                   0367 +2300090371
03      X             SR    0                                                             0368 -5000000000
04      X             B     2+X93                                              00009      0369 +0193090002
                      LITERALS
05      X                   +5000000000                                              09   0370 +5000000000   0370
06      X                   +9999999999                                              09   0371 +9999999999   0371
07      X                   +0                                                       00   0372 +0            03
08      X                   +11                                                      12   0372 + 11          0.
```

```
PAGE AA     PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                                  CDNO FD  LOC  INSTRUCTION    REF

01 1520   *                ARITH EXAMPLE 7
02 1521              DA    1                                                                       +0003250327
03 1522   A                00,07A4.4                                                     07  0325                0325
04 1523   B                08,15A4.4                                                     89  0325                0325
05 1524   X                20,29F                                                        09  0327                0327
06 1525   *
07 1526   ANYLABEL   ARITH X=A-B+15/(A-B)                                                                         M
08       X ANYLABEL  ZS2   B(0,1)                                                   00001    0328 -2300890325
09       X           SL2   2                                                                 0329 +5000002202
10       X           S2    B(2,3)                                                            0330 -2400010326
11       X           SL2   4                                                                 0331 +5000002204
12       X           A2    A(0,7)                                                            0332 +2400070325
13       X           ST2   COMAREA.A(0,9)+1                                         00002    0333 +2200090392
14       X           ZA2   +15                                                               0334 +2300340396
15       X           ZA3   COMAREA.A(0,9)+1                                                  0335 +3300090392
16       X           BLX   93,DIV1.A                                                         0336 +0200930351
17       X           XA    MACREG.1,0+1                                                      0337 +4700010001
18       X           B     DIV2.A                                                   00003    0338 +0100090359
19       X           ST2   COMAREA.A(0,9)+1                                                  0339 +2200090392
20       X           ZS2   B(0,1)                                                            0340 -2300890325
21       X           SL2   2                                                                 0341 +5000002202
22       X           S2    B(2,3)                                                            0342 -2400010326
23       X           SL2   4                                                        00004    0343 +5000002204
24       X           A2    A(0,7)                                                            0344 +2400070325
25       X           A2    COMAREA.A(0,9)+1                                                  0345 +2400090392
26       X           ZA1   +0                                                                0346 +1300000396
27       X           ZA3   +0000000070                                                       0347 +3300090393
28       X           BLX   94,FLOT2.A                                               00005    0348 +0200940385
29       X           ST1   X(0,9)                                                            0349 +1200090327
30 1527   *
31 15281  *         THE FOLLOWING IS GENERATED OUT OF LINE
32 15282  *
33       X           B     DIV3.A+6 TEMP BRANCH FOR GENOS                                    0350 +0100090383
34       X DIV1.A    BZ3   DIV3.A                                                            0351 +3000090377
35       X           ZA1   +0                                                                0352 +1300000396
         X           SLC   MACREG.1                                                 00006    0353 -5000010300

                                                                                             0387 -3400450001
24       X           STD3  9991(0,1)                                                00013    0388 +5000001002
25       X FLOT3.A   B     0+X94                                                             0389 -3200019991
26       X COMAREA.A DA                                                                      0390 +0194090000
                     LITERALS                                                                     +0003910392
27       X                 +0000000070                                              00014 09 0393 +0000000070    0393
28       X                 +5000000000                                                    09 0394 +5000000000    0394
29       X                 +9999999999                                                    09 0395 +9999999999    0395
30       X                 +0                                                             00 0396 +0             0396
31       X                 +11                                                            12 0396 + 11           0396
32       X                 +15                                                            34 0396 +   15         0396
```

```
ERROR MESSAGE LIST

PG/LN    MESSAGE

AA 07 ARITH W 20 0404
```

ARITH Example 7

Warning message W 20 has been produced because the division operation might develop more integer digits than can be accommodated by the mask 4.4, which has been included in the message in the form 0404.

```
PAGE AA     PROGRAM                                                  7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP    OPERAND                                                       CDNO FD  LOC  INSTRUCTION      REF

01  101   *                 ARITH EXAMPLE 8
02  102               DA    1                                                                            +0003250334
03  103   A                 03,19A9.8                                                          39  0325                    0325
04  104   B                 37,49A9.4                                                          79  0328                    0328
05  105   C                 53,69A9.8                                                          39  0330                    0330
06  106   D                 72,79A5.3                                                          29  0332                    0332
07  107   RESULT            80,99A12.8                                                         09  0333                    0333
08  108   *
09  109   ANYLABEL    ARITH RESULT=(A+B-ABS(C))*D/A                                                                          M
10      X ANYLABEL    ZA1   C(0,6)                                                        00001      0335 +1300390330
11      X             ZA2   C(7,16)                                                                  0336 +2300090331
12      X             MSP   9991                                                                     0337 -0300919991
13      X             MSP   9992                                                                     0338 -0300919992
14      X             ST1   COMAREA.A(0,9)+2                                                         0339 +1200090511
15      X             ST2   COMAREA.A(0,9)+3                                              00002      0340 +2200090512
16      X             ZA1   +0                                                                       0341 +1300000516
17      X             ZA2   B(0,2)                                                                   0342 +2300790328
18      X             SL    1                                                                        0343 -5000000201
19      X             A2    B(3,3)                                                                   0344 +2400000329
20      X             BLX   94,LINK.A                                                     00003      0345 +0200940372
21      X             SL    8                                                                        0346 -5000000208
22      X             A1    A(0,6)                                                                   0347 +1400390325
23      X             A2    A(7,16)                                                                  0348 +2400090326
24      X             BLX   94,LINK.A                                                                0349 +0200940372
25      X             S1    COMAREA.A(0,9)+2                                              00004      0350 -1400090511
26      X             S2    COMAREA.A(10,19)+2                                                       0351 -2400090512
27      X             BLX   94,LINK.A                                                                0352 +0200940372
28      X             ST1   COMAREA.A(0,9)                                                           0353 +1200090509
29      X             ST2   COMAREA.A(10,19)                                                         0354 +2200090510
30      X             ZA1   +0                                                            00005      0355 +1300000516
31      X             ZA2   D(0,7)                                                                   0356 +2300290332
32      X             BLX   94,LINK.A                                                                0357 +0200940372
33      X             BLX   93,DUBLMPY.A                                                             0358 +0200930395
34      X             XS    MACREG.1,0+17                                                            0359 -4700010017
35      X             B     DUBLCK.A                                                      00006      0360 +0100090431
36      X             ST1   COMAREA.A(0,9)                                                           0361 +1200090509
37      X             ST2   COMAREA.A(10,19)                                                         0362 +2200090510
38      X             ZA1   A(0,6)                                                                   0363 +1300390325
39      X             ZA2   A(7,16)                                                                  0364 +2300090326
40      X             BLX   94,LINK.A                                                     00007      0365 +0200940372
41      X             BLX   93,DUBLDIV.A                                                             0366 +0200930453
42      X             XA    MACREG.1,0+11                                                            0367 +4700010011
43      X             B     DUBLCK.A                                                                 0368 +0100090431
44      X             ST1   RESULT(0,9)                                                              0369 +1200090333
45      X             ST2   RESULT(10,19)                                                 00008      0370 +2200090334
46  110   *
47  111   *         THE FOLLOWING IS GENERATED OUT OF LINE
48      X             B     LINK8.A+1                                                                0371 +0100090394
```

```
PAGE AB     PROGRAM                                                                                          PAGE AB

LN CDREF LABEL        OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION         REF

01       X LINK.A     BV2    LINK1.A IS THERE A CARRY                                0372 +2100090380
02       X LINK3.A    BZ2    LINK4.A                                                 0373 +2000090377
03       X            BZ1    LINK5.A                                                 0374 +1000090378
04       X            BM1    LINK6.A                                        00009    0375 -1000090385
05       X            BM2    LINK7.A                                                 0376 -2000090390
06       X LINK4.A    SR     0 USE SIGN OF 9991                                      0377 -5000000000
07       X LINK5.A    SL     0 USE SIGN OF 9992                                      0378 -5000000200
08       X            B      0+X94                                                   0379 +0194090000
09       X LINK1.A    BM2    LINK2.A                                        00010    0380 -2000090383
10       X            A1     +1 POSITIVE CARRY                                       0381 +1400110516
11       X            B      LINK3.A                                                 0382 +0100090373
12       X LINK2.A    S1     +1 NEGATIVE CARRY                                       0383 -1400110516
13       X            B      LINK3.A                                                 0384 +0100090373
14       X LINK6.A    BM2    0+X94 ARE BOTH ACC MINUS                       00011    0385 -2094090000
15       X            A1     +1 REVERSE CARRY (NEG)                                  0386 +1400110516
16       X            S2     +9999999999 COMPLEMENT + SIGNG)                         0387 -2400090515
17       X            S2     +1 CHANGE    9992  NG)                                  0388 -2400110516
18       X            B      0+X94                                                   0389 +0194090000
19       X LINK7.A    S1     +1 REVERSE CARRY (POS)                         00012    0390 -1400110516
20       X            A2     +9999999999 COMPLEMENT + SIGNS)                         0391 +2400090515
21       X            A2     +1 CHANGE 9992                                          0392 +2400110516
22       X LINK8.A    B      0+X94                                                   0393 +0194090000
23       X            B      HOLD.A+3 TEMP BRANCH                                    0394 +0100090430
24       X DUBLMPY.A  SLC    MACREG.1                                       00013    0395 -5000010300
25       X            ZST1   MULT.A                                                  0396 -1100090425
26       X            ZST2   MULT.A+1                                                0397 -2100090426
27       X            ZA1    COMAREA.A                                               0398 +1300090509
28       X            ZA2    COMAREA.A+1                                             0399 +2300090510
29       X            SLC    MACREG.2                                       00014    0400 -5000020300
30       X            XA     MACREG.1,0+MACREG.2                                     0401 +4702010000
31       X            ZST1   HOLD.A                                                  0402 -1100090427
         X            ZA3    9992                                                    0403 +3300099992

35       X            A3     +5000000000
36       X            B      DUBLOVFL.A+2
37       X            DA     1                                                            +0005060508
38       X DSOR.A            00,19                                             09  0506                    0506
39       X QUOT.A            20,29                                             09  0508                    0508
40       X COMAREA.A  DA                                                                  +0005090512
                      LITERALS
41       X                   +0000010009                                00035  09  0513 +0000010009    0513
42       X                   +5000000000                                       09  0514 +5000000000    0514
43       X                   +9999999999                                       09  0515 +9999999999    0515
44       X                   +0                                                00  0516 +0             0516
45       X                   +1                                                11  0516 + 1            0516
46       X                   +20                                               23  0516 +  20          0516
```

ERROR MESSAGE LIST

PG/LN MESSAGE

AA 09 ARITH W 20 1208

ARITH Example 8

Warning message W 20 has been produced because a multiplication or division operation might develop more integer digits than can be accommodated by the mask 12.8, which has been included in the message in the form 1208.

COMP—Compare and Branch

COMP generates instructions to compare two fields and to branch according to the results of the comparison.

Source Program Format

The basic format for the COMP statement in the source program is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–)
01	ANYLABEL	COMP	FIELD1, FIELD2, LOWBR, EQUBR, HIGHBR
02			

ANYLABEL may be any symbolic label; it may be omitted. The entry COMP must be written exactly as shown. FIELD1 and FIELD2 are either the symbolic names of the fields to be compared, or alphameric or numerical literals. Adcons are not permitted. LOWBR, EQUBR, and HIGHBR are the symbolic labels of instructions to which the program will branch if FIELD1 has the following relation to FIELD2, depending on the mode:

MODE	*LOWBR*	*EQUBR*	*HIGHBR*
Numerical	is less than	equals	is greater than
Alphameric	precedes	is identical to	follows

These results are determined by comparison techniques described under "Processing Techniques," below.

The basic format may be varied in two ways:

1. If the comparison is between numerical fields, either one or both may be replaced by the expression ABS(FIELDX), in which case the absolute value of the field will be used for the comparison. The parentheses must be written as shown. An attempt to take the absolute value of an alphameric field will result in an error message during assembly.
2. One or two of the branch addresses may be omitted. Instead of the missing branch, the object program would then take the next instruction. If another branch is specified after an omitted branch, separating commas must be punched; e.g., FIELD1, FIELD2, LOWBR, , HIGHBR.

Processing Techniques

All comparisons are made on the basis of the standard IBM 7070 collating sequence. The fields to be compared may be numerical (either automatic-decimal or floating-decimal) or alphameric.

Limitations on Length

The number of parameters is limited by the format. Automatic-decimal fields to be compared may bridge words but may not exceed twenty digits in length. Floating-decimal numbers must be contained within one location. Numerical

literals may not exceed twenty digits in length and must be signed. There is no limit on the length of alphameric fields, except that alphameric literals may not exceed 120 characters.

Address Modification

Modification by indexing and address adjustment is permitted on all symbolic addresses.

The Effect of COMP

Comparison of numerical fields is accomplished by subtraction, the correct branch address being determined by a negative, zero, or positive difference.

When an automatic-decimal number is to be compared to a floating-decimal number, the automatic-decimal number is first converted to floating-decimal format.

For the comparison of two automatic-decimal fields, the alignment of decimal points is automatic. If, after alignment of decimal points, the total number of integer and decimal digits in both fields is larger than 20, the excess number of digits will be truncated on the right without rounding; no warning message will be issued. A difference between the numbers in the truncated digits would not register in the comparison. Thus, if the following fields were to be compared using the COMP macro-instruction, the transfer would be to EQUBR:

Fields	*Format Specifications*	*Object Program Contents*
FIELD1	A12.8	000000000001.23456789
FIELD2	A2.18	01.234567891000000000

If an automatic-decimal field of more than twenty digits in length is compared to a floating-decimal field, an automatic-decimal field, or an alphameric field, a warning message will be issued. Instructions will be generated to compare the absolute values of the fields.

If a numerical field (either automatic-decimal or floating-decimal), which is of proper length, is compared to an alphameric field, a warning message will be issued. Instructions will be generated to compare the absolute values of the fields.

Fields containing numbers that are in double-digit representation must be converted to single-digit form before the COMP statement is employed (e.g., by use of the EDMOV macro-instruction). But numerical fields of different modes may be compared to each other, as may fields of the various alphameric types.

It should be noted that the COMP macro-instruction treats all fields according to their defined characteristics (or the absence of them), and not according to their object-program contents. Thus, difficulties may arise if numerical data is stored in fields defined as alphameric, or vice versa.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

ALL BRANCHES BLANK

All branches have been omitted from a comparison in which one or both of the fields are absolute. The comparison is made; three branches will be generated to the next in-line instructions (*, *+1, and *+2) for patching purposes.

ALL BRANCHES EQUAL

All three branches are identical; coding will be generated.

FIELD 1 BLANK

Field 1 has been omitted. A NOP will be generated.

FIELD 2 BLANK

Field 2 has been omitted. A NOP will be generated.

FIELD 1 NOT ACCEPTABLE

Field 1 is an adcon. A NOP will be generated.

FIELD 2 NOT ACCEPTABLE

Field 2 is an adcon. A NOP will be generated.

LESS THAN 3 INPUT PARAMETERS

The minimum input in the COMP statement is FIELD1, FIELD2, and a branch address. The above message is issued if this minimum requirement is not met. A NOP will be generated.

NUMERIC FIELD GREATER THAN 20 DIGITS

A numerical field is greater than 20 digits in length. Instructions will be generated to compare the absolute values of the fields.

W-BOTH FIELDS NOT ALPHA-NOFORM

Either one field is alphameric and the other is not or one field is unspecified ("noform") and the other is not. Instructions will be generated to compare the absolute values of the fields.

W-UNUSUAL BRANCH CONDITION

A branch address in the operand of the COMP statement has *not* been left blank or is an address *other* than the label of an imperative statement (symbolic machine instruction or macro-instruction), or an actual storage address.

Examples

The following are examples of acceptable coding for the COMP macro-instruction. For each, the associated source-program entries are given, followed by the COMP statement, coding generated in-line, and (where applicable) coding generated out-of-line.

```
PAGE AA    PROGRAM                                      7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                                   CDNO FD   LOC  INSTRUCTION     REF

01  115   *                COMP EXAMPLE  1.
02  116              DA    1                                                                          +0003250328
03  117   FIELD1           02,19A8.10                                                       29   0325                  0325
04  118   FIELD2           23,39A8.9                                                        39   0327                  0327
05  119   LOWBR      NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE         00001      0329 -0100090000
06  120   EQUALBR    NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                    0330 -0100090000
07  121   HIGHBR     NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                    0331 -0100090000
08  122   *
09  123   ANYLABEL   COMP  FIELD1,FIELD2,LOWBR,EQUALBR,HIGHBR
10      X ANYLABEL   ZS1   FIELD2(0,6)                                                           0332 -1300390327
11      X            ZS2   FIELD2(7,16)                                                          0333 -2300090328
12      X            BLX   94,LINK.A                                                   00002     0334 +0200940345
13      X            SL    1                                                                     0335 -5000000201
14      X            A1    FIELD1(0,7)                                                           0336 +1400290325
15      X            A2    FIELD1(8,17)                                                          0337 +2400090326
16      X            BLX   94,LINK.A                                                             0338 +0200940345
17      X            BZ2   *+2                                                         00003     0339 +2000090341
18      X            B     *+2                                                                   0340 +0100090342
19      X            BZ1   EQUALBR                                                               0341 +1000090330
20      X            BM1   LOWBR                                                                 0342 -1000090329
21      X            B     HIGHBR                                                                0343 +0100090331
22  124   *
23  125   *        THE FOLLOWING IS GENERATED OUT OF LINE
24  126   *
25      X            B     LINK8.A+1                                                   00004     0344 +0100090367
26      X LINK.A     BV2   LINK1.A IS THERE A CARRY                                              0345 +2100090353
27      X LINK3.A    BZ2   LINK4.A                                                               0346 +2000090350

                     A2    +1 CHANGE 9992                                                        0365 +2400000368
47      X LINK8.A    B     0+X94                                                                 0366 +0194090000
                     LITERALS
48      X                  +9999999999                                                      09   0367 +9999999999   0367

                           OPERAND

01      X                  +1                                                               00   0368 +1            0368
```

COMP Example 1

The program will branch to LOWBR if FIELD1 is less than FIELD2, to EQUALBR if they are equal, and to HIGHBR if FIELD1 is greater than FIELD2.

```
PAGE AA    PROGRAM                                   7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                              CDNO FD  LOC  INSTRUCTION     REF

01  130   *                 COMP EXAMPLE 2
02  131              DA     1                                                                 +0003250328
03  132   FIELD1            02,19A8.10                                                 29  0325                 0325
04  133   FIELD2            23,39A8.9                                                  39  0327                 0327
05  134   EQUALBR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE   00001     0329 -0100090000
06  135   *
07  136   ANYLABEL   COMP   FIELD1,FIELD2,,EQUALBR
08      X ANYLABEL   ZS1    FIELD2(0,6)                                                     0330 -1300390327
09      X            ZS2    FIELD2(7,16)                                                    0331 -2300090328
10      X            BLX    94,LINK.A                                                       0332 +0200940341
11      X            SL     1                                                               0333 -5000000201
12      X            A1     FIELD1(0,7)                                       00002         0334 +1400290325
13      X            A2     FIELD1(8,17)                                                    0335 +2400090326
14      X            BLX    94,LINK.A                                                       0336 +0200940341
15      X            BZ2    *+2                                                             0337 +2000090339
16      X            B      *+2                                                             0338 +0100090340
17      X            BZ1    EQUALBR                                           00003         0339 +1000090329
18  137   *
19  138   *       THE FOLLOWING IS GENERATED OUT OF LINE
20  139   *
21      X            B      LINK8.A+1                                                       0340 +0100090363
22      X LINK.A     BV2    LINK1.A IS THERE A CARRY                                        0341 +2100090349
23      X LINK3.A    BZ2    LINK4.A                                                         0342 +2000090346
24      X            BZ1    LINK5.A                                                         0343 +1000090347
25      X            BM1    LINK6.A                                           00004         0344 -1000090354
26      X            BM2    LINK7.A                                                         0345 -2000090359
27      X LINK4.A    SR     0 USE SIGN OF 9991                                              0346 -5000000000
28      X LINK5.A    SL     0 USE SIGN OF 9992                                              0347 -5000000200
29      X            B      0+X94                                                           0348 +0194090000
30      X LINK1.A    BM2    LINK2.A                                           00005         0349 -2000090352
31      X            A1     +1 POSITIVE CARRY                                               0350 +1400000364
32      X            B      LINK3.A                                                         0351 +0100090342
33      X LINK2.A    S1     +1 NEGATIVE CARRY                                               0352 -1400000364
34      X            B      LINK3.A                                                         0353 +0100090342
35      X LINK6.A    BM2    0+X94 ARE BOTH ACC MINUS                          00006         0354 -2094090000
36      X            A1     +1 REVERSE CARRY (NEG)                                          0355 +1400000364
37      X            S2     +9999999999 COMPLEMENT + SIGNG)                                 0356 -2400090363
38      X            S2     +1 CHANGE    9992  NG)                                          0357 -2400000364
39      X            B      0+X94                                                           0358 +0194090000
40      X LINK7.A    S1     +1 REVERSE CARRY (POS)                            00007         0359 -1400000364
41      X            A2     +9999999999 COMPLEMENT + SIGNS)                                 0360 +2400090363
42      X            A2     +1 CHANGE 9992                                                  0361 +2400000364
43      X LINK8.A    B      0+X94                                                           0362 +0194090000
                     LITERALS
44      X                   +9999999999                                            09  0363 +9999999999     0363
45      X                   +1                                            00008 00  0364 +1               0364
```

COMP Example 2

This example is the same as Example 1 except that both LOWBR and HIGHBR have been omitted. The program will continue sequentially if FIELD1 is either greater than or less than FIELD2.

```
PAGE AA     PROGRAM                                     7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP    OPERAND                                                  CDNO FD  LOC  INSTRUCTION    REF

01  142   *                 COMP EXAMPLE 3.
02  143               DA    1
                                                                                               +0003250326
03  144   FIELD1            02,19A8.10
04  145               DA    2,RDW,0+INDEXWORD                                            29  0325                 0325
05         X                                                                                   +0003270336
06         X                                                                       00001     0327 +0003290332   0327
07  146   FIELD2            23,39A8.9                                                        0328 -0003330336   0328
08  147   LOWBR       NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE         39  0331                 0002
09  148   HIGHBR      NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE    00002     0337 -0100090000
10  149   *                                                                                  0338 -0100090000
11  150   ANYLABEL    COMP  ABS(FIELD1),FIELD2,LOWBR,,HIGHBR
12         X ANYLABEL ZA1   FIELD1(0,7)
13         X          ZA2   FIELD1(8,17)                                                     0339 +1300290325
14         X          MSP   9991                                                             0340 +2300090326
15         X          MSP   9992                                                             0341 -0300919991
16         X          ST1   COMAREA.A(0,9)+2                                       00003     0342 -0300919992
17         X          ST2   COMAREA.A(0,9)+3                                                 0343 +1200090382
18         X          ZS1   FIELD2(0,6)+INDEXWORD                                            0344 +2200090383
19         X          ZS2   FIELD2(7,16)+INDEXWORD                                           0345 -1301390002
20         X          BLX   94,LINK.A                                                        0346 -2301090003
21         X          SL    1                                                      00004     0347 +0200940358
22         X          A1    COMAREA.A(0,9)+2                                                 0348 -5000000201
23         X          A2    COMAREA.A(10,19)+2                                               0349 +1400090382
24         X          BLX   94,LINK.A                                                        0350 +2400090383
25         X          BZ2   *+2                                                              0351 +0200940358
26         X          B     *+2                                                    00005     0352 +2000090354
27         X          BZ1   *+3                                                              0353 +0100090355
28         X          BM1   LOWBR                                                            0354 +1000090357
29         X          B     HIGHBR                                                           0355 -1000090337
30  151   *                                                                                  0356 +0100090338
31  152   *           THE FOLLOWING IS GENERATED OUT OF LINE
32  153   *
33         X          B     LINK8.A+1
34         X LINK.A   BV2   LINK1.A IS THERE A CARRY                               00006     0357 +0100090380
35         X LINK3.A  BZ2   LINK4.A                                                          0358 +2100090366
36         X                                                                                 0359 +2000090363
                                                                                             0360 +1000090364
                                                                                             0361 -1000090371

03         X          B     0+X94                                                            0374 -2400000385
04         X LINK7.A  S1    +1 REVERSE CARRY (POS)                                           0375 +0194090000
05         X          A2    +9999999999 COMPLEMENT + SIGNS)                                  0376 -1400000385
06         X          A2    +1 CHANGE 9992                                         00010     0377 +2400090384
07         X LINK8.A  B     0+X94                                                            0378 +2400000385
08         X COMAREA.A DA                                                                    0379 +0194090000
                      LITERALS                                                                    +0003800383
09         X                +9999999999
10         X                +1                                                     00011 09  0384 +9999999999   0384
                                                                                         00  0385 +1            0385
```

COMP Example 3

The program will branch to LOWBR if the absolute value of FIELD1 is less than FIELD2 and to HIGHBR if it is greater. If the absolute value of FIELD1 equals FIELD2, the program will continue sequen'' .ly.

```
PAGE AA     PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION      REF

01  157   *                 COMP EXAMPLE 4.
02  158              DA     2,,0+INDEXWORD                                               +0003250328
03  159   FIELD1            00,19'                                               09  0325                  0000
04  160              DA     1                                                            +0003290332
05  161   FIELD2            20,39'                                               09  0331                  0331
06  162   LOWBR      NOP          REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001     0333 -0100090000
07  163   EQUALBR    NOP          REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE            0334 -0100090000
08  164   HIGHBR     NOP          REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE            0335 -0100090000
09  165   *
10  166   ANYLABEL   COMP   FIELD1,FIELD2,LOWBR,EQUALBR,HIGHBR
11      X ANYLABEL   ZAA    FIELD2(0,9)                                                 0336 +1600090331
12      X            CA     FIELD1(0,9)+INDEXWORD                                       0337 -1501090000
13      X            BL     HIGHBR                                         00002        0338 +4000090335
14      X            BH     LOWBR                                                       0339 -4000090333
15      X            ZAA    FIELD2(10,19)                                               0340 +1600090332
16      X            CA     FIELD1(10,19)+INDEXWORD                                     0341 -1501090001
17      X            BL     HIGHBR                                                      0342 +4000090335
18      X            BE     EQUALBR                                        00003        0343 -4100090334
19      X            B      LOWBR                                                       0344 +0100090333
20  167   *
```

COMP Example 4

If FIELD1 precedes, is identical to, or follows FIELD2 in the standard collating sequence, the program will branch to LOWBR, EQUALBR or HIGHBR, respectively.

```
PAGE AA     PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION      REF

01  171   *                 COMP EXAMPLE 5.
02  172              DA     2,,0+X15                                                     +0003250328
03  173   FIELD1            00,19'                                               09  0325                  0000
04  174              DA     2,,0+INDEXWORD                                               +0003290338
05  175   FIELD2            20,45'                                               09  0331                  0002
06  176   DIFFBR     NOP          REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001     0339 -0100090000
07  177   EQUBR      NOP          REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE            0340 -0100090000
08  178   *
09  179   ANYLABEL   COMP   FIELD1,FIELD2,DIFFBR,EQUBR,DIFFBR
10      X ANYLABEL   ZAA    FIELD1(0,9)+X15                                             0341 +1615090000
11      X            CA     FIELD2(0,9)+INDEXWORD                                       0342 -1501090002
12      X            BL     DIFFBR                                                      0343 +4000090339
13      X            BH     DIFFBR                                         00002        0344 -4000090339
14      X            ZAA    FIELD1(10,19)+X15                                           0345 +1615090001
15      X            CA     FIELD2(10,19)+INDEXWORD                                     0346 -1501090003
16      X            BL     DIFFBR                                                      0347 +4000090339
17      X            BH     DIFFBR                                                      0348 -4000090339
18      X            ZA1    FIELD2(0,9)+2+INDEXWORD                        00003        0349 +1301090004
19      X            BZ1    EQUBR                                                       0350 +1000090340
20      X            B      DIFFBR                                                      0351 +0100090339
21  180   *
```

COMP Example 5

Fields which are identical will cause a branch to EQUBR; fields which differ to DIFFBR.

```
PAGE  AA     PROGRAM                                                      7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN  CDREF   LABEL       OP      OPERAND                                                   CDNO  FD   LOC  INSTRUCTION      REF

01   184    *                   COMP EXAMPLE 6.
02   185                DA      1                                                                         +0003250326
03   186    FIELD1              00,09F                                                           09   0325                   0325
04   187    FIELD2              10,19F                                                           09   0326                   0326
05   188    EQUALBR     NOP          REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE     00001       0327 -0100090000
06   189    HIGHBR      NOP          REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                 0328 -0100090000
07   190    *
08   191    ANYLABEL    COMP    FIELD1,FIELD2,,EQUALBR,HIGHBR
09        X ANYLABEL    ZA1     FIELD1(0,9)                                                            0329 +1300090325
10        X             S1      FIELD2(0,9)                                                            0330 -1400090326
11        X             BV1     *+2                                                                    0331 +1100090333
12        X             BZ1     EQUALBR                                                   00002        0332 +1000090327
13        X             BM1     *+2                                                                    0333 -1000090335
14        X             B       HIGHBR                                                                 0334 +0100090328
15   192    *
```

COMP Example 6

The program will go to the next sequential instruction if FIELD1 is less than FIELD2, to EQUALBR if FIELD1 is equal to FIELD2, and to HIGHBR if FIELD1 is greater than FIELD2.

```
PAGE  AA     PROGRAM                                                      7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN  CDREF   LABEL       OP      OPERAND                                                   CDNO  FD   LOC  INSTRUCTION      REF

01   201    *                   COMP EXAMPLE 7.
02   202                DA      1                                                                         +0003250332
03   203    FIELD1              00,39'                                                           09   0325                   0325
04   204    FIELD2              40,79'                                                           09   0329                   0329
05   205    DIFFBR      NOP          REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE     00001       0333 -0100090000
06   206    *
07   207    ANYLABEL    COMP    FIELD1,FIELD2,DIFFBR,,DIFFBR
08        X ANYLABEL    XL      MACREG.01,+0000000003                                                  0334 +4500010340
09        X M.4         ZAA     FIELD2(0,9)+MACREG.01                                                  0335 +1601090329
10        X             CA      FIELD1(0,9)+MACREG.01                                                  0336 -1501090325
11        X             BL      DIFFBR                                                                 0337 +4000090333
12        X             BH      DIFFBR                                                    00002        0338 -4000090333
13        X             BIX     MACREG.01,M.4                                                          0339 +4900010335
14   208    *
                        LITERALS
15        X                     +0000000003                                                      09   0340 +0000000003       0340
```

COMP Example 7

If FIELD1 = FIELD2, the program will continue with the next sequential instruction. If not, it will branch to DIFFBR.

```
PAGE AA    PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA
LN CDREF LABEL      OP    OPERAND                                                    CDNO FD  LOC  INSTRUCTION    REF
01  214  *                COMP EXAMPLE 8.
02  215             DA    1                                                                        +0003250327
03  216  FIELD1           00,09F                                                          09  0325                0325
04  217  FIELD2           10,29A13.7                                                      09  0326                0326
05  218  LOWBR      NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE         00001      0328 -0100090000
06  219  EQUALBR    NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                    0329 -0100090000
07  220  *
08  221  ANYLABEL   COMP  FIELD1,FIELD2,LOWBR,EQUALBR
09      X ANYLABEL  ZA1   FIELD2(0,9)                                                           0330 +1300090326
10      X           ZA2   FIELD2(10,19)                                                         0331 +2300090327
11      X           ZA3   +0000000063                                                           0332 +3300090347
12      X           BLX   94,FLOT2.A                                                 00002      0333 +0200940341
13      X           S1    FIELD1(0,9)                                                           0334 -1400090325
14      X           BV1   *+2                                                                   0335 +1100090337
15      X           BZ1   EQUALBR                                                               0336 +1000090329
16      X           BM1   *+2                                                                   0337 -1000090339
17      X           B     LOWBR                                                      00003      0338 +0100090328
18  222  *
19  223  *        THE FOLLOWING IS GENERATED OUT OF LINE
20  224  *
21      X FLOT1.A   SLC1  MACREG.1                                                              0339 +5000011300
22      X           B     *+2                                                                   0340 +0100090342
23      X FLOT2.A   SLC   MACREG.1                                                              0341 -5000010300
24      X           BZ1   0+X94                                                                 0342 +1094090000
25      X           S3    MACREG.1(4,5)                                              00004      0343 -3400450001
26      X           SR1   2                                                                     0344 +5000001002
27      X           STD3  9991(0,1)                                                             0345 -3200019991
28      X FLOT3.A   B     0+X94                                                                 0346 +0194090000
                    LITERALS
29      X                 +0000000063                                                     09  0347 +0000000063    0347
```

COMP Example 8

The program will continue with the next instruction if FIELD1 is greater than FIELD2. It will branch to EQUALBR if FIELD1 equals FIELD2 and to LOWBR if FIELD1 is less than FIELD2.

```
PAGE AA    PROGRAM                                  7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL       OP    OPERAND                                                CDNO FD  LOC  INSTRUCTION    REF

01  227   *                       COMP EXAMPLE 9                                                 +0003250336
02  228   AREANAME    DA    3,,0+INDEXWORD                                                29  0325                 0000
03  229   FIELD1            02,19A8.10                                                    39  0327                 0002
04  230   FIELD2            23,39A8.9                                              00001      0337 -0100090000
05  231   LOWBR       NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE              0338 -0100090000
06  232   EQUALBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE              0339 -0100090000
07  233   HIGHBR      NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE
08  234   *
09  235   ANYLABEL    COMP  FIELD1,FIELD2,LOWBR,        REMARKS MAY
10  236                     EQUALBR,HIGHBR              BE USED.                              0340 -1301390002
11         X ANYLABEL ZS1   FIELD2(0,6)+INDEXWORD                                             0341 -2301090003
12         X          ZS2   FIELD2(7,16)+INDEXWORD                                 00002      0342 +0200940353
13         X          BLX   94,LINK.A                                                         0343 -5000000201
14         X          SL    1                                                                 0344 +1401290000
15         X          A1    FIELD1(0,7)+INDEXWORD                                             0345 +2401090001
16         X          A2    FIELD1(8,17)+INDEXWORD                                            0346 +0200940353
17         X          BLX   94,LINK.A                                              00003      0347 +2000090349
18         X          BZ2   *+2                                                               0348 +0100090350
19         X          B     *+2                                                               0349 +1000090338
20         X          BZ1   EQUALBR                                                           0350 -1000090337
21         X          BM1   LOWBR                                                             0351 +0100090339
22         X          B     HIGHBR
23  237    *
24  238    *          THE FOLLOWING IS GENERATED OUT OF LINE
25  239    *                                                                       00004      0352 +0100090375
26         X          B     LINK8.A+1                                                         0353 +2100090361
27         X LINK.A   BV2   LINK1.A IS THERE A CARRY                                          0354 +2000090358
28         X LINK3.A  BZ2   LINK4.A                                                           0355 +1000090359
29         X          BZ1   LINK5.A                                                           0356 -1000090366
30         X          BM1   LINK6.A                                                00005      0357 -2000090371
31         X          BM2   LINK7.A                                                           0358 -5000000000
32         X LINK4.A  SR    0 USE SIGN OF 9991                                                0359 -5000000200
33         X LINK5.A  SL    0 USE SIGN OF 9992                                                0360 +0194090000
34         X          B     0+X94                                                             0361 -2000090364
35         X LINK1.A  BM2   LINK2.A

                                                                                              0370 +0194090000
45         X LINK7.A  S1    +1 REVERSE CARRY (POS)                                            0371 -1400000376
46         X          A2    +9999999999 COMPLEMENT + SIGNS)                        00008      0372 +2400090375
47         X          A2    +1 CHANGE 9992                                                    0373 +2400000376
48         X LINK8.A  B     0+X94                                                             0374 +0194090000

                      LITERALS
01         X                +9999999999                                               09  0375 +9999999999     0375
02         X                +1                                                        00  0376 +1              0376
```

CYCLE—Cycle Branch
RECYC—Reset Cycle

CYCLE generates instructions to branch a specified number of times to each of a series of locations. RECYC generates instructions to reinitialize one or more CYCLE macro-instructions.

Source Program Format

The basic formats for the CYCLE and RECYC statements in the source program are as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–55) Basic Autocoder
01	ANYLABEL	CYCLE	BRANCH1, COUNTER1, BRANCH2, COUNTER2, etc.
02	ANYLABEL	RECYC	CYCLE1, CYCLE2, etc.
03			

In these examples, ANYLABEL is any symbolic label; it may be omitted. The entries CYCLE and RECYC must be written exactly as shown. In the CYCLE statement, the various BRANCHX entries are the symbolic labels of instructions to which the program will transfer, taking each BRANCH from left-to-right and going to each as many times as is specified by the associated COUNTERX. The branching will occur every time the program reaches the position originally occupied by the CYCLE statement. After each BRANCH is taken, it is assumed that the program will return to a location preceding the location of the CYCLE statement or to that location itself. The COUNTERS may be symbolic or they may be unsigned actual integers (representing literal count numbers) up to a maximum of ten digits. If the reference is symbolic, the count number must be stored in a single location; i.e., it may not bridge words. Symbolic and literal counters may be freely mixed within the operand of a single CYCLE macro-instruction.

After all BRANCHES have been executed the number of times indicated, the next pass through the CYCLE statement will resume the entire process from the beginning. However, if there is only one BRANCH and one COUNTER, the macro-instruction becomes inactive (in effect, a NOP) after the branch has been made the requisite number of times, and the program continues sequentially.

The CYCLEX entries in the operand portion of the RECYC statement are the labels of CYCLE macro-instructions. The RECYC macro-instruction will restore the settings of the various counters in these CYCLE statements to their original values. Thus, after a CYCLE statement has been named in a RECYC macro-instruction, the next program pass through that CYCLE instruction will result in a "first-time" transfer to the BRANCH1 address.

The format of the CYCLE instruction may be modified in two ways:

1. A BRANCHX may be omitted, with separating commas entered as in the following example:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	CYCLE	BR1,,CTR1,,,CTR2
02			

   If this is done, the program will go to the next instruction of the source program for the specified number of times, instead of to the missing BRANCHX address. Thereafter, the next indicated BRANCH will be taken as usual. In particular, it is possible to omit the BRANCH1, beginning the operand portion of the CYCLE statement with a comma. In that case, the CYCLE statement will, in effect, result in a NOP on the first n program passes, where n is the number specified by the COUNTER1.

2. The last COUNTER may be omitted as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	CYCLE	BR1,,CTR1,,BR2,,CTR2,,BR3
02			

   This will cause the program to branch permanently to the last BRANCH after the other BRANCHES have been executed the required number of times. These two devices may be combined in a single CYCLE statement; for an example, see "Limitations on Length," below.

If the counters are symbolic, the number of times the program is to branch to each location must have been entered into the proper field before the set of instructions generated by CYCLE is entered for the first time. When the program reaches the CYCLE instruction, all COUNTERS in the operand are locked; they cannot be altered until all BRANCHES have been executed as required, except through a RECYC macro-instruction. Since the BRANCHES are taken from left-to-right, it is possible to alter the COUNTER for a *prior* BRANCH while a given BRANCH is being taken. This change will take effect when the entire CYCLE is restarted.

Processing Techniques

Limitations on Length

A maximum of five BRANCH and COUNTER pairs, including omitted parameters, may be entered. If more than five pairs are required, two successive CYCLE macro-instructions may be written as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–55) — Basic Autocoder
01	ANYLABEL1	CYCLE	BR1,,CTR1,,BR2,,CTR2,,BR3,,CTR3,,BR4,,CTR4,,,
02	ANYLABEL2	CYCLE	BR5,,CTR5,,BR6,,CTR6,,etc.
03			

The first CYCLE statement will become inactive after the fourth BRANCH has been taken the required number of times; the program will then permanently "fall through" to the second statement.

RECYC allows for 94 entries in the operand portion.

Address Modification

All symbolic addresses may be modified by indexing and address adjustment.

Error and Warning Messages

The CYCLE macro generator will issue the following error and warning messages during assembly under the conditions specified:

ASSUME COMMAS AFTER PAR. 2 — PERMANENT NOP

This warning message is issued if the operand contains only one BRANCH and one COUNTER. After the BRANCH has been taken the specified number of times, the macro-instruction will not be reinitialized (which would make it the equivalent of an unconditional Branch) but will become permanently inactive (NOP). This is what would happen if the COUNTER were followed by two commas.

BRIDGE CTR USING FIRST LOC ONLY PAR *xx*

A symbol COUNTER has been used that bridges two storage locations. The message will give the parameter number of the faulty entry in place of the *xx*. Only the digits from the portion in the first location will be used as the COUNTER.

ERROR — IMPROPER OPERAND

This message is issued if there are less than two or more than ten parameters in the operand, or if an omitted COUNTER is followed by another BRANCH, which would never become effective.

WARNING — SUCCESSIVE NOPS

This message will be issued if two successive BRANCHES have been omitted from the operand. Nevertheless, the appropriate coding will be generated as if the double omission were intentional.

The RECYC macro generator will issue the following error message under the condition specified:

PARAM NOT LABEL OF CYCLE MACRO *xx*

The *xx* will be replaced by the number of the parameter which is not the label of a CYCLE statement.

Examples

The following are examples of acceptable coding for the CYCLE and RECYC macro-instructions. For each, the associated source-program entries are given, followed by the CYCLE or RECYC statement, coding generated in-line, and (where applicable) coding generated out-of-line.

```
PAGE AA     PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                               CDNO FD  LOC   INSTRUCTION      REF

01  243   *                 CYCLE EXAMPLE 1.
02  244   BRANCHA    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE   00001    0325 -0100090000
03  245   BRANCHB    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE            0326 -0100090000
04  246   *
05  247   ANYLABEL   CYCLE  BRANCHA,1,BRANCHB,1
06      X ANYLABEL   NOP    M.1                                                            0327 -0100090330
07      X            MSP    ANYLABEL                                                       0328 -0300910327
08      X            B      BRANCHA                                                        0329 +0100090325
09      X M.1        MSM    ANYLABEL                                              00002    0330 -0300610327
10      X            B      BRANCHB                                                        0331 +0100090326
11      X M.6        NOP    0                                                              0332 -0100090000
12      X ORIGIN     CNTRL  *-1
13  248   *
```

CYCLE Example 1

CYCLE is used in this example to create a strictly alternating condition.

The first time the program arrives at this instruction, it will branch to BRANCHA, the next time to BRANCHB, then to BRANCHA, then to BRANCHB again, and so on for the duration of the program.

```
PAGE AA     PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                               CDNO FD  LOC   INSTRUCTION      REF

01  252   *                 CYCLE EXAMPLE 2.
02  253   POINTX     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE   00001    0325 -0100090000
03  254   POINTY     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE            0326 -0100090000
04  255   *
05  256   ANYFIELD   CYCLE  POINTX,1,POINTY
06      X ANYFIELD   NOP    M.1                                                            0327 -0100090330
07      X            MSP    ANYFIELD                                                       0328 -0300910327
08      X            B      POINTX                                                         0329 +0100090325
09      X M.1        B      POINTY                                                00002    0330 +0100090326
10      X M.6        NOP    0                                                              0331 -0100090000
11      X ORIGIN     CNTRL  *-1
12  257   *
```

CYCLE Example 2

When the program arrives at CYCLE the first time, it will branch to POINTX. Each subsequent time it will branch to POINTY.

```
PAGE AA     PROGRAM                                  7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA
LN CDREF  LABEL        OP    OPERAND                                         CDNO FD  LOC  INSTRUCTION    REF
01  261   *                  CYCLE EXAMPLE 3.
02  262   TOTALLINE    NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001     0325 -0100090000
03  263   REPORTLINE   NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE            0326 -0100090000
04  264   *
05  265   ANYLABEL     CYCLE REPORTLINE,35,TOTALLINE,1
06       X ANYLABEL    NOP   M.1                                                         0327 -0100090331
07       X             ZSA   +35                                                         0328 -1600010338
08       X             ST1   M.9(2,5)                                                    0329 +1200250337
09       X             MSP   ANYLABEL                                        00002       0330 -0300910327
10       X M.1         XL    MACREG.1,M.9                                                0331 +4500010337
11       X             XA    MACREG.1,1                                                  0332 +4700010001
12       X             XU    MACREG.1,M.9                                                0333 -4500010337
13       X             BXM   MACREG.1,REPORTLINE                                         0334 -4400010326
14       X             MSM   ANYLABEL                                        00003       0335 -0300610327
15       X             B     TOTALLINE                                                   0336 +0100090325
16  266   *
17  267   *         THE FOLLOWING IS GENERATED OUT OF LINE
18  268   *
19       X M.9         DA    1                                                                +0003370337
20       X                   00,09                                               09  0337                0337
                           LITERALS
21       X                   +35                                             00004 01  0338 +35          0338
```

CYCLE Example 3

In this example, the program will branch to the REPORTLINE routine 35 times, to the TOTALLINE routine once, then to the REPORTLINE routine 35 times, to the TOTALLINE routine once, and so on.

```
PAGE AA     PROGRAM                                  7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA
LN CDREF  LABEL        OP    OPERAND                                         CDNO FD  LOC  INSTRUCTION    REF
01  272   *                  CYCLE EXAMPLE 4.
02  273   BRANCHONE    NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001     0325 -0100090000
03  274   BRANCHTWO    NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE            0326 -0100090000
04  275   *
05  276   ANYLABEL     CYCLE BRANCHONE,1,,1,BRANCHTWO
06       X ANYLABEL    NOP   M.1                                                         0327 -0100090331
07       X             MSP   ANYLABEL                                                    0328 -0300910327
08       X             MSM   M.1                                                         0329 -0300610331
09       X             B     BRANCHONE                                       00002       0330 +0100090325
10       X M.1         NOP   M.2                                                         0331 -0100090334
11       X             MSP   M.1                                                         0332 -0300910331
12       X             B     M.6                                                         0333 +0100090335
13       X M.2         B     BRANCHTWO                                                   0334 +0100090326
14       X M.6         NOP   0                                               00003       0335 -0100090000
15       X ORIGIN      CNTRL *-1
16  277   *
```

CYCLE Example 4

The first time the program reaches CYCLE, a branch is made to BRANCHONE. On the second pass the next instruction following those generated by CYCLE is executed, since a branch address was omitted. Each subsequent time, a branch is made to BRANCHTWO since the final counter was omitted.

```
PAGE AA    PROGRAM                                        7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                               CDNO FD  LOC  INSTRUCTION     REF

01  301   *                  CYCLE EXAMPLE 5.
02  302   BRANCH1     NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE   00001     0325 -0100090000
03  303   BRANCH3     NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE             0326 -0100090000
04  304   COUNTER1    NOP        REPRESENTS COUNTER LABEL                                   0327 -0100090000
05  305   COUNTER2    NOP        REPRESENTS COUNTER LABEL                                   0328 -0100090000
06  306   COUNTER3    NOP        REPRESENTS COUNTER LABEL                                   0329 -0100090000
07  307   *
08  308   ANYLABEL    CYCLE  BRANCH1,COUNTER1,,COUNTER2
09      X ANYLABEL    NOP    M.1                                                  00002     0330 -0100090338
10      X             ZSA    COUNTER1                                                       0331 -1600090327
11      X             ST1    M.9+1                                                          0332 +1200090350
12      X             ZSA    COUNTER2                                                       0333 -1600090328
13      X             ST1    M.9+2                                                          0334 +1200090351
14      X             ZAA    M.8                                                  00003     0335 +1600090365
15      X             ST1    M.9                                                            0336 +1200090349
16      X             MSP    ANYLABEL                                                       0337 -0300910330
17      X M.1         XL     MACREG.1,M.9                                                   0338 +4500010349
18      X M.2         ZA1    M.9+MACREG.1                                                   0339 +1301090349
19      X             A1     +1                                                   00004     0340 +1400000367
20      X             ZST1   M.9+MACREG.1                                                   0341 -1101090349
21      X             BM1    M.5-1+MACREG.1                                                 0342 -1001090351
22      X             XL     MACREG.1,M.9                                                   0343 +4500010349
23      X             XA     MACREG.1,1                                                     0344 +4700010001
24      X             XU     MACREG.1,M.9                                         00005     0345 -4500010349
25      X             BCX    MACREG.1,M.2                                                   0346 -4300010339
26      X             MSM    ANYLABEL                                                       0347 -0300610330
27      X             B      ANYLABEL                                                       0348 +0100090330
28      X M.9         DA     1                                                                   +0003490351
29      X                    00,29                                                   09     0349                  0349
30      X M.5         B      BRANCH1                                              00006     0352 +0100090325
31      X             B      M.6                                                            0353 +0100090354
32      X M.6         NOP    0                                                              0354 -0100090000
33      X ORIGIN      CNTRL  *-1
34  309               CYCLE  BRANCH3,COUNTER3,,1        ***
35      X M.19        NOP    M.10                                                           0354 -0100090358
36      X             ZSA    COUNTER3                                                       0355 -1600090329
37      X             ST1    M.18(2,5)                                                      0356 +1200250366
38      X             MSP     M.19                                                00007     0357 -0300910354
39      X M.10        XL     MACREG.1,M.18                                                  0358 +4500010366
40      X             XA     MACREG.1,1                                                     0359 +4700010001
41      X             XU     MACREG.1,M.18                                                  0360 -4500010366
42      X             BXM    MACREG.1,BRANCH3                                               0361 -4400010326
43      X             MSM     M.19                                                00008     0362 -0300610354
44      X M.15        NOP    0                                                              0363 -0100090000
45      X ORIGIN      CNTRL  *-1
46  310   SOMELABEL   RECYC  ANYLABEL                   ***
47      X SOMELABEL   MSM    ANYLABEL                                                       0363 -0300610330
48  311               B      ANYLABEL                   ***                                 0364 +0100090330
```

```
PAGE AB     PROGRAM                                                                          PAGE AB

LN CDREF  LABEL       OP     OPERAND                                    CDNO FD  LOC  INSTRUCTION    REF

01  312   *
02  313   *       THE FOLLOWING IS GENERATED OUT OF LINE
03  314   *
04      X M.8         DC                                                            +0003650365
05      X                    +0000010002                                   09  0365 +0000010002    0365
06      X M.18        DA     1                                                      +0003660366
07      X                    00,09                                         09  0366                0366
                    LITERALS
08      X                    +1                                      00009 00  0367 +1             0367
```

CYCLE Example 5

This example illustrates a technique that will allow the modification of the contents of a counter associated with a branch when one of the preceding branches has already been entered. In this case, COUNTER3 may be freely changed while BRANCH1 is being taken by the program. (This would not be the case if COUNTER3 were written into the operand of the same CYCLE statement as BRANCH1.) COUNTER2 must be set to a value greater than the maximum possible content of COUNTER3. This must be done before the first CYCLE statement is entered. The program will BRANCH to BRANCH1 as many times as specified in COUNTER1. It will then "fall through" to the second CYCLE statement, taking BRANCH3 as many times as COUNTER3 indicates, and finally "fall through" to the RECYC statement. This will reinitialize the first CYCLE macro-instruction (the second one will have reinitialized itself, having completed all branches the required number of times), to which a transfer is then made by means of the unconditional Branch instruction.

NOTE: Lines marked thus *** in the example listing are original source statements; the intervening instructions are generated.

```
PAGE AA     PROGRAM                               7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001, PAGE AA

LN CDREF  LABEL      OP     OPERAND                                    CDNO FD  LOC  INSTRUCTION    REF

01  318   *                 RECYC EXAMPLE 1.
02  319   SOMELABEL  RECYC  ANYLABEL,LABELZ,ANYSYMBOL,POINTX
03      X SOMELABEL  MSM    ANYLABEL                                  00001      0325 -0300610329
04      X            MSM    LABELZ                                               0326 -0300610353
05      X            MSM    ANYSYMBOL                                            0327 -0300610377
06      X            MSM    POINTX                                               0328 -0300610401
07  320   *
08  321   ANYLABEL   CYCLE  BRANCH1,COUNTER1,,COUNTER2
09      X ANYLABEL   NOP    M.1                                                  0329 -0100090337
10      X            ZSA    COUNTER1                                  00002      0330 -1600090426
11      X            ST1    M.9+1                                                0331 +1200090349
12      X            ZSA    COUNTER2                                             0332 -1600090427
13      X            ST1    M.9+2                                                0333 +1200090350
14      X            ZAA    M.8                                                  0334 +1600090428
```

RECYC Example 1

The CYCLE macro-instructions whose labels are listed in the operand are reinitialized by this instruction.

DECOD—Branch on Code Value

DECOD generates instructions to analyze a code field and to branch according to the value it contains. The code field must have been previously established through a CODE declarative statement.

Source Program Format

The basic format for the DECOD statement in the source program is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–60) Basic Autocoder →	Autocoder (61–70)
01	ANYLABEL	DECOD	CODENAME, CODEVALUE1, BRANCH1, CODEVALUE2, B	RANCH2, etc.
02				

ANYLABEL is any symbolic label; it may be omitted. The entry DECOD must be written exactly as shown. CODENAME must be the symbolic label of a CODE header line that appears elsewhere in the program. The CODEVALUEX entries are the symbolic names of the conditions for which tests are to be made; they must be subsequent entries under the CODE header line, CODENAME. The BRANCHX entries are the symbolic addresses of instructions to which the program is to transfer if the associated CODEVALUE is present in the CODE field to be analyzed.

Omission of BRANCH addresses is not permitted in writing the DECOD statement. If it is desired to have the program continue sequentially in case a given CODEVALUE is present in the CODENAME field, the particular CODEVALUE should be omitted from the operand altogether.

Processing Techniques

Limitations on Length

A maximum of 45 CODEVALUE and BRANCH pairs may be entered.

Address Modification

The CODENAME and BRANCH entries may be modified by indexing and address adjustment.

Processing Sequence

The CODENAME field is analyzed by the generated instructions for the presence of the various CODEVALUES as they appear in the operand portion of the source statement from left to right. Thus, if two different labels of CODE subsequent entries refer to the same actual value in the CODE field, and if both are listed, with different BRANCHES, in the operand of a single DECOD statement, the presence of that value in the CODENAME field will cause a transfer to the first of these BRANCHES.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

CODENAME NOT DEFINED BY A CODE

This error message is issued if the first parameter is not the label of a CODE statement.

CODEVALUE NOT DEFINED UNDER A CODE

This error message is issued if one of the CODEVALUE entries is not a subsequent entry under any CODE header line.

CODEVALUES AND BRANCHES NOT PAIRED

This message will be issued if the number of parameters following CODENAME is odd.

WARNING — CODEVALUE NOT DEFINED UNDER CODENAME

This warning message is issued if one of the CODEVALUE entries has, as its first parameter record, a header label which is different from the CODENAME written in the source statement. Coding will be generated nevertheless.

Examples

The following are examples of acceptable coding for the DECOD statement. For each, the associated source-program entries are given, followed by the DECOD statement, coding generated in-line, and coding generated out-of-line.

```
PAGE AA     PROGRAM                                   7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01  332   *                  DECOD EXAMPLE 1.
02  333   AREANAME    DA     5,,0+X30                                                       +0003250329
03  334   CODENAME    CODE   6,6                                                   66  0325                 0000
04  335    CODEVALUE         1
05  336   BRANCH1     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001    0330 -0100090000
06  337   *
07  338   ANYLABEL    DECOD  CODENAME,CODEVALUE,BRANCH1
08      X ANYLABEL    ZA1    CODENAME+X30                                              0331 +1330660000
09      X                    CA     +1                                                 0332 -1500000335
10      X             BE     BRANCH1                                                   0333 -4100090330
11  339   *
12  340   *        THE FOLLOWING IS GENERATED OUT OF LINE
13  341   *
                      LITERALS
14      X                    1                                                 00  0334 +1              0334
15      X                    +1                                          00002 00  0335 +1              0335
```

DECOD Example 1

The program will branch to BRANCH1 if the CODENAME field contains the condition specified for CODEVALUE, or will go to the next sequential instruction if it does not.

```
PAGE AA     PROGRAM                                   7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01  345   *                  DECOD EXAMPLE 2.
02  346               DA     1                                                              +0003250325
03  347   CODENAME    CODE   0,9'                                                  09  0325                 0325
04  348    CODE1             'RED  '
05  349    CODE2             'GREEN'
06  350    CODE3             'BLUE '
07  351   BR1         NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001    0326 -0100090000
08  352   BR2         NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE           0327 -0100090000
09  353   BR3         NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE           0328 -0100090000
10  354   *
11  355   ANYLABEL    DECOD  CODENAME,CODE1,BR1,CODE2,BR2,
12  356                      CODE3,BR3
13      X ANYLABEL    ZA1    CODENAME                                                  0329 +1300090325
14      X             CA     RED                                                       0330 -1500090341
15      X             BE     BR1                                              00002    0331 -4100090326
16      X             CA     GREEN                                                     0332 -1500090340
17      X             BE     BR2                                                       0333 -4100090327
18      X             CA     BLUE                                                      0334 -1500090339
19      X             BE     BR3                                                       0335 -4100090328
20  357   *
21  358   *        THE FOLLOWING IS GENERATED OUT OF LINE
22  359   *
                      LITERALS
23      X                    'RED  '                                     00003 09  0336 '7965640000     0336
24      X                    'GREEN'                                           09  0337 '6779656575     0337
25      X                    'BLUE '                                           09  0338 '6273846500     0338
```

DECOD Example 2

If the CODENAME field contains the value specified for CODE1, CODE2, or CODE3, the program will branch to BR1, BR2, or BR3, respectively. Otherwise the program will go to the next sequential instruction.

LOGIC generates instructions to test whether a given expression is true or false and, according to the result, to set a switch or to branch to a designated location, or both.

Source Program Format

The basic formats for the LOGIC statement in the source program are as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50 Basic Au
01	ANYLABEL	LOGIC	SWITCH=EXPRESSION
02	ANYLABEL	LOGIC	EXPRESSION, TRUEBR, FALSEBR
03	ANYLABEL	LOGIC	SWITCH=EXPRESSION, TRUEBR, FALSEBR
04			

In these examples, ANYLABEL is any symbolic label; it may be omitted. The entry LOGIC must be written exactly as shown. SWITCH represents any electronic, program, or digit switch to be turned ON if the expression in question is true, OFF if false. If the label of an index word or a DA, DC, or DSW header line or subsequent entry is written, the digits in the first location of the current area referenced will be treated as a bank of digit switches; i.e., they will all be turned ON or OFF. TRUEBR and FALSEBR represent the symbolic labels of instructions to which the program will branch if the expression is, respectively, true or false.

The first format causes the object program to set the switch as indicated and to continue with the next instruction of the source program after the LOGIC statement. The second format causes the object program to branch only, not to set a switch. The third format does both.

In the second and third formats listed above, one of the branches may be omitted. The object program will then proceed sequentially instead of taking the missing branch. For example, if the operand reads SWITCH=EXPRESSION, TRUEBR and the expression is false, the designated switch will be turned OFF and the next instruction of the source program will be executed. Care must be taken to enter the separating commas if TRUEBR is omitted since the generator always interprets the branch following the first comma as TRUEBR. The operand should read SWITCH=EXPRESSION, ,FALSEBR.

The table on the following page is a list of the parameters which are valid in the LOGIC statement as well as the way they are treated by LOGIC when they are used as a SWITCH or as an EXPRESSION.

Logical Expressions

Logical expressions, in their simplest form, may consist of a single *logical variable*. *Logical operators* are used to connect two or more logical variables in order to form more complex logical expressions. *Logical punctuation* is used when necessary to clarify an ambiguous logical expression.

Parameter	Switch	Expression
Index Word	Bank of Switches	Operand Field
Alteration Switch	INVALID	Alteration Switch
Electronic Switch Undefined Term	Electronic Switch	Electronic Switch
DLINE Header Line DLINE Subsequent Entry CODE Header Line	INVALID	Operand Field
CODE Subsequent Entry	INVALID	Code Field
DTF Header Line	Bank of Switches	Operand Field
DA Header Line DC Header Line DSW Header Line	Digit Switch or Bank of Switches	Operand Field
DTF Subsequent Entry DA Subsequent Entry DC Subsequent Entry	Digit Switch or Bank of Switches	Digit Switch or Operand Field
DSW Subsequent Entry	Digit Switch	Digit Switch
Program Switch	Program Switch	Program Switch

Also, literals are allowed in relational expressions. Any parameters not listed above are invalid.

Logical variables may be of seven types; they can assume either one of two values, which are interpreted as true and false, respectively.

Type	*Address*	*True Condition*	*False Condition*
Electronic Switch	Symbolic	ON	OFF
Alteration Switch	"	ON	OFF
Program Switch	"	Plus	Minus
Digit Switch	"	>0	=0
Code Value	"	Code value present	Code value not present
Operand Field	"	Not all zeros	All zeros
Relational	(See below)	True	False

Electronic Switch, Alteration Switch, Program Switch, Digit Switch. These four switches require no explanation. Examples of the use of these switches with the LOGIC macro-instruction are included under "Examples." The use of an electronic

switch is illustrated in examples 6, 9, and 10; of an alteration switch in examples 7 and 10; of a program switch in examples 7, 9, and 10; of a digit switch in examples 1, 8, and 10.

Code Values. If a CODE field has been defined in accordance with the instructions outlined on page 50, it can be interrogated for the presence of a specific code value by referencing the symbolic name of that code value. If the 50 states of the Union are assigned the integers 1 through 50 in alphabetical order as code values under a CODE header line labeled STATE, the integer 1 will be assigned to ALABAMA. The word ALABAMA then functions as a logical variable, and in any given expression it will be regarded as true if the CODE field STATE contains a 1, false otherwise. The use of a code value is illustrated in examples 5 and 10 under "Examples."

Operand Fields. An operand field is a contiguous field of any length, not necessarily confined to a single word. If an operand field is used as a SWITCH to be set, it is treated as a bank of switches. However, if the field is greater than one word, only the first word of the field will be turned on and a warning message will be issued.

If an operand field is used as an EXPRESSION in the LOGIC statement, it may be any length. The field will be regarded as false if it contains all zeros, and true if one digit contains a value other than zero. The sign of the zeros does not matter. Alphameric *zeros* (@9090909090) will register as digits different from zero, but alphameric *blanks* (@0000000000) will be treated as zeros in this context. The use of an operand field as an EXPRESSION is illustrated in example 3 under "Examples."

Relationals. Relational expressions are comparisons between two numerical or two alphameric fields; the two types should not be mixed in a single comparison.

The six comparison operators available and their respective numerical and alphameric meanings are as follows:

Operator	*Numerical Meaning*	*Alphameric Meaning*
G	is greater than	follows
NOT G	is not greater than	does not follow
E	is equal to	is identical to
NOT E	is not equal to	is not identical to
L	is less than	precedes
NOT L	is not less than	does not precede

A relational expression is formed by placing one of these operators between two fields and enclosing the resulting expression in parentheses. Care must be taken that exactly one blank separates fields and the operator.

When comparing two numerical fields, either field may be symbolic or literal. Numerical literals may be signed or unsigned; if unsigned, they will be interpreted as positive. Both automatic-decimal and floating-decimal fields are acceptable, and the modes may be freely mixed within a single relational expression.

The conditions and restrictions which apply to the use of the LOGIC macro-instruction are identical to those which apply to the COMP macro-instruction. These are as follows:

1. No automatic-decimal field may have more than twenty digits.
2. If an automatic-decimal number is compared to a floating-decimal number, only its first eight significant digits will affect the comparison.
3. If two automatic-decimal numbers are compared, their combined length after alignment of the decimal point may not exceed twenty digits. Any excess decimals beyond this length are disregarded in the comparison.

Comparison of numerical fields is strictly algebraic. Of two positive fields, the one with the greater absolute value will be regarded as larger of two negative numbers, the one with the smaller absolute value will be regarded as larger; any negative number is treated as smaller than any positive; zeros, whether positive or negative, will be treated as smaller than any other positive and greater than any other negative number.

For example, (AGE G 17) will be true if the number in the field referenced by AGE is larger than +17, false otherwise. The expression (TEMP NOT L − 13) is true if TEMP is 2, 0, −5, or −13, false if it is −13.5. Examples 2, 8, and 10 under "Examples" illustrate additional uses of relationals for comparing numerical fields.

When comparing two *alphameric fields,* either field may be symbolic or literal. Literals must be enclosed in @ signs (e.g., @NYC@). There is no limitation on the length of alphameric fields to be compared, except that literals may not exceed 120 characters. The relational expression tests for dictionary ordering, and the comparison operators must be reinterpreted to the alphameric meaning indicated in the chart above. Special characters will be included in this dictionary ordering according to the standard collating sequence given in the IBM Reference Manual "7070 Data Processing System."

For example, (INITIAL L @K@) will be regarded as true whenever the field referenced by the word INITIAL contains a letter that precedes K in the alphabet, or a special character whose two-digit numerical representation is less than 72. (GRADE NOT G @X@) will be false if the GRADE field contains Y or Z, true if it contains V, W, or X. (@12@ G @AB@) is true. @ @ will be regarded as less than any other alpha field.

Logical Operators

Logical operators permit the construction of more complex expressions from logical variables. The LOGIC macro generator interprets three operators: NOT, AND, and OR.

Not. If NOT precedes an expression, it has the effect of changing its value to the opposite; if it precedes a true expression, the resulting expression is false, and vice versa. The use of NOT is illustrated in examples 6, 9, and 10 under "Examples."

And. If AND is placed between two expressions, a new and more complex expression results. This expression is true if, and only if, both of the component expressions are true; otherwise it is false. The use of AND is illustrated in examples 7, 9, and 10 under "Examples."

Or. If OR is placed between two expressions, a new and more complex expression results. This expression is true if at least one of the component expressions is true, possibly both. The compound expression, therefore, is false only if both of the components are false. The use of OR is illustrated in example 8 and 10 under "Examples."

Logical Punctuation (Parentheses)

Expressions resulting from logical operations upon variables may in turn serve as components for larger expressions. Let capital letters represent electronic

switches; then the expression A OR NOT B will be true if A is on, or if NOT B is true, i.e., if B is off, or both. It will be false if and only if A is off and B is on.

The expression NOT A AND B, however, is ambiguous as it stands. It might have been constructed from A and B by first operating upon them with AND, and then prefacing the result with NOT; in that case the expression will be true if B is off, regardless of the status of A. On the other hand, it might have been built by placing NOT before A, and then operating upon the resulting expression and B with the operator AND; in that case, if B is off the expression is false. To provide the necessary distinction between these meanings, any compound logical expression must be enclosed in parentheses before it is operated upon again. This would yield NOT (A AND B) and (NOT A) AND B, respectively, for the two cases above. The use of parentheses is illustrated in examples 9 and 10 under "Examples."

To avoid excessive parenthesization, three conventions are adopted:

1. The operator NOT applies only to the shortest complete logical expression immeditaely to its right. Thus NOT A OR B is taken to mean (NOT A) OR B, since A is a complete logical expression. If NOT (A OR B) was intended, the parentheses would have to be explicitly written; then, since the operator NOT is followed by a left parenthesis, the shortest complete expression to its right is the entire parenthesized expression.
2. If the operators AND and OR occur in the same expression without intervening parentheses, the terms connected by AND will be understood to be parenthesized. Thus A AND B OR C will be taken to mean (A AND B) OR C. If A AND (B OR C) is intended, the parentheses must be written.
3. Repeated use of either AND or OR in the same logical expression is not ambiguous and need not be parenthesized. The expression A OR B OR C always yields the same result, whether treated as (A OR B) OR C or as A OR (B OR C). However, because of a saving in object program time, LOGIC will deal with it as though it had the latter form. See "Left-Orientation," below.

The LOGIC macro generator will automatically interpret punctuated logical expressions in the sense of these conventions. It will not, however, reject clear, correct expressions in which parentheses are explicit that might have been suppressed.

Processing Techniques

Limitations on Length

Not more than 24 parameters may occur in any one LOGIC operand, counting the switch to be set, each branch, each logical variable, and each logical operator; in the case of repetitions, each occurrence is counted separately. A relational expression without NOT has three parameters; with NOT, four. Punctuation and address modifiers are not counted as parameters.

Spacing and Punctuation

A blank must both precede and follow each AND and OR. If NOT precedes a parenthesized expression, no blank need intervene, but if it precedes some other operand, a space should be left blank. In relational expressions, no blanks should occur between the enclosing parentheses and the fields to be compared. The operator should be separated by one blank from each of the fields, and, where NOT appears, from each other. No blanks should occur on either side of the equal sign or on either side of the separating commas preceding branches.

Address Modification

Addresses occurring in the operand portion of a LOGIC statement may be freely modified by indexing and address adjustment. The same is true of symbolic

fields in relational expressions though not of literal fields in the same expressions; this will, of course, require parentheses within parentheses. Address modification is illustrated in example 10 under "Examples."

Left-Orientation

In programming complex logical expressions, placing the simpler terms or conditions on the left side of AND and OR operators will often result in a substantial saving of object program time. Thus, the logical expression

A OR NOT B AND (AGE G 26) (1)

will yield coding which may allow faster resolution than the logically equivalent version

(AGE G 26) AND NOT B OR A (2)

In the object program, the truth of the individual logical variables is evaluated from left to right in the order in which they appeared in the source statement. Thus, if A is true in a given run of the object program, the above expression as a whole is true, and if it was originally coded as in version (1), the determination of the truth of B and of (AGE G 26) is bypassed. In version (2), no such quick resolution can be obtained. (AGE G 26) will be the first logical variable to be interrogated. The outcome of this test would not constitute sufficient information for an evaluation of the truth of the expression as a whole. If the relational expression is true, B would have to be tested next, and if false, A.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

EQUAL SIGN BEGINS INPUT — WILL IGNORE

This warning will be issued if the operand portion of the source statement begins with an equal sign. The sign will be disregarded, and no instructions will be generated to set a switch *unless* a second equal sign occurs in the statement following the first operand entry.

ILLEGAL PUNCTUATION MARK USED

The only punctuation that may validly appear in a LOGIC statement consists of an equal sign and a maximum of two commas as indicated under "Source Program Format," as well as parentheses, and plus, minus, and alpha signs. Other special characters may occur only inside alphameric literals. In case of violation, a NOP will be generated.

ILLEGAL TERM ENDS LOGICAL EXPRESSION

Some term that is neither a logical variable nor a right parenthesis ends the source statement, or immediately precedes the first separating comma before the branch entries. This condition may be caused if an unintended double blank is interpreted by the generator to mean that the operand portion is complete. A NOP will be generated.

ILLEGAL TERM PRECEDES A BINARY OPERATOR

A term that is neither a right parenthesis nor a logical variable precedes an AND or OR. A NOP will be generated.

ILLEGAL TERM PRECEDES A LEFT PAREN

Some term that is neither another left parenthesis, nor a logical operator, nor an equal sign, precedes a left parenthesis. A NOP will be generated.

ILLEGAL TERM PRECEDES A NOT

A term that is neither a logical operator, nor a left parenthesis, nor an equal sign, precedes a NOT. A NOP will be generated.

ILLEGAL TERM PRECEDES A RIGHT PAREN

Some term that is neither another right parenthesis, nor a logical variable, nor an address modifier, precedes a right parenthesis. A NOP will be generated.

ILLEGAL TERM PRECEDES PARAMETER *xx*

xx will be replaced by the number of a parameter which has been preceded by a term that is neither a logical operator, nor a left parenthesis, nor an equal sign. A NOP will be generated.

INVALID PARAMETER *xx*

This message will be issued if a parameter is not one specifically listed as valid under "Source Program Format." *xx* will be replaced by the number of the parameter at fault. A NOP will be generated.

NO BRANCH OR SWITCH TO BE SET IN INPUT

The coding fails to indicate what implementation the generated instructions are to initiate. A NOP will be generated.

NOTHING TO TEST IN LOGIC STATEMENT

The source statement does not contain an expression whose truth or falsity is to be determined. A NOP is generated.

PAREN. MISSING AROUND ARITH-REL

Either the left or the right parenthesis has been omitted around a relational expression. This message will also be issued if an attempt is made to run together more than one comparison since the generator expects to find a right parenthesis after the second field. For example, this message will be issued if (MIN L AVERAGE L MAX) is encountered since the generator expects to find a right parenthesis after AVERAGE.

PARENTHESIS NOT CLOSED

There is an excess of left parentheses, leaving at least one left parenthesis unpaired. A NOP will be generated.

TOO MANY RIGHT PARENTHESES

There is an excess of right parentheses, leaving at least one right parenthesis unpaired. A NOP will be generated.

WILL SET SWITCHES IN FIRST LOCATION ONLY

This warning is issued if the SWITCH to be set is the label of a DA or DC header line or subsequent entry, and if the referenced area bridges words. Only those digits that lie in the first location will be affected by the generated instructions.

Examples

The following are examples of acceptable coding for the LOGIC macro-instruction. For each, the associated source-program entries are given, followed by the LOGIC statement, coding generated in-line, and (where applicable) coding generated out-of-line.

```
PAGE AA    PROGRAM                                            7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                                CDNO FD  LOC  INSTRUCTION    REF

01  363   *                LOGIC EXAMPLE 1.
02  364              DSW   DIGITSW                                               00001 00  0325 +1000000000    0325
03  365   FALSEBR    NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE              0326 -0100090000
04  366   *
05  367   ANYLABEL   LOGIC DIGITSW,,FALSEBR
06      X ANYLABEL   CD    DIGITSW,0                                                       0327 +0300000325
07      X            BE    FALSEBR                                                         0328 -4100090326
08  368   *
```

LOGIC Example 1

If the digit switch is ON, the program will continue with the next instruction.

If it is OFF, it will branch to FALSEBR.

```
PAGE AA    PROGRAM                                            7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                                CDNO FD  LOC  INSTRUCTION    REF

01  372   *                LOGIC EXAMPLE 2.
02  373              DA    1                                                                    +0003250325
03  374   INCOME           0,4A                                                           04  0325                0325
04  375   TRUEBR     NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE    00001       0326 -0100090000
05  376   *
06  377   ANYLABEL   LOGIC SWITCH=(INCOME G 4800),TRUEBR,
07      X ANYLABEL   ZA2   INCOME(0,4)                                                    0327 +2300040325
08      X            S2    +4800                                                          0328 -2400030334
09      X            BZ2   M.2                                                            0329 +2000090333
10      X            BM2   M.2                                                            0330 -2000090333
11      X M.1        ESN   SWITCH                                             00002       0331 +6100100000
12      X            B     TRUEBR                                                         0332 +0100090326
13      X M.2        ESF   SWITCH                                                         0333 +6100200000
14  378   *
15  379   *        THE FOLLOWING IS GENERATED OUT OF LINE
16  380   *
                     LITERALS
17      X                  +4800                                                      03  0334 +4800          0334
```

LOGIC Example 2

If the numerical field referenced by INCOME is greater than +4800, SWITCH will be turned ON and the program will branch to TRUEBR. If INCOME is equal to or less than +4800, SWITCH will be turned OFF and the program will continue sequentially. In this example, since the type of switch is not specifically designated, SWITCH is assumed to be electronic switch 1.

```
PAGE AA     PROGRAM                                         7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                              CDNO FD  LOC  INSTRUCTION     REF

01  384   *                 LOGIC EXAMPLE 3.
02  385              DA     1                                                                 +0003250325
03  386   OPFIELD           0,9A                                                      09  0325                 0325
04  387   TRUEBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001     0326 -0100090000
05  388   *
06  389   ANYLABEL   LOGIC  OPFIELD,TRUEBR
07      X ANYLABEL   ZA1    OPFIELD(0,9)                                                  0327 +1300090325
08      X            BZ1    M.1                                                           0328 +1000090330
09      X            B      TRUEBR                                                        0329 +0100090326
10      X M.1        NOP                                                                  0330 -0100000000
11      X ORIGIN     CNTRL  *-1
12  390   *
```

LOGIC Example 3

If OPFIELD contains any digits other than zeros, the program will continue with the instruction labeled TRUEBR. If every digit is zero, the program continues sequentially. Alphameric blanks (@0000000000) register as zeros; alphameric zeros (@9090909090) register as digits other than zero.

```
PAGE AA     PROGRAM                                         7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                              CDNO FD  LOC  INSTRUCTION     REF

01  401   *                 LOGIC EXAMPLE 4.
02  402              DA     1                                                                 +0003250325
03  403   SUBSCRIPT         5,6'                                                      56  0325                 0325
04  404   FALSEBR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001     0326 -0100090000
05  405   *
06  406   ANYLABEL   LOGIC  SWITCH=(SUBSCRIPT NOT E 'J'),,FALSEBR
07      X ANYLABEL   ZAA    M.5(0,1)                                                      0327 +1600010334
08      X            CA     SUBSCRIPT(0,1)                                                0328 -1500560325
09      X            BL     M.1                                                           0329 +4000090333
10      X            BH     M.1                                                           0330 -4000090333
11      X M.2        ESF    SWITCH                                               00002     0331 +6100200000
12      X            B      FALSEBR                                                       0332 +0100090326
13      X M.1        ESN    SWITCH                                                        0333 +6100100000
14  407   *
15  408   *       THE FOLLOWING IS GENERATED OUT OF LINE
16  409   *
17      X            DC                                                                   +0003340334
18      X M.5               'J'                                                  00003 01  0334 '71            0334
```

LOGIC Example 4

If the field referenced by SUBSCRIPT, which must be alphameric, contains a J, SWITCH will be turned OFF and the program will branch to FALSEBR. If SUBSCRIPT contains a character different from J, SWITCH will be turned ON and the program will continue sequentially.

```
PAGE AA     PROGRAM                                        7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                                  CDNO FD  LOC  INSTRUCTION     REF

01  413   *                 LOGIC EXAMPLE 5.
02  414              DA     1                                                                      +0003250325
03  415   STATE      CODE   0                                                             00  0325                  0325
04  416    OHIO             5
05  417   TRUEBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE   00001       0326 -0100090000
06  418   FALSEBR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                0327 -0100090000
07  419   *
08  420   ANYLABEL   LOGIC  SWITCH=OHIO,TRUEBR,FALSEBR
09      X ANYLABEL   CD     STATE,5                                                            0328 +0300500325
10      X            BE     M.1                                                                0329 -4100090332
11      X M.2        ESF    SWITCH                                                             0330 +6100200000
12      X            B      FALSEBR                                                  00002     0331 +0100090327
13      X M.1        ESN    SWITCH                                                             0332 +6100100000
14      X            B      TRUEBR                                                             0333 +0100090326
15  421   *
16  422   *       THE FOLLOWING IS GENERATED OUT OF LINE
17  423   *
                     LITERALS
18      X                   5                                                         00  0334 +5                   0334
```

LOGIC Example 5

Assume that a CODE declarative statement has established a one-digit CODE field labeled STATE, and that the code value corresponding to OHIO is 5. If the STATE field contains a 5, SWITCH will be turned ON and the program will branch to TRUEBR; otherwise, SWITCH will be turned OFF and the program will branch to FALSEBR.

```
PAGE AA     PROGRAM                                        7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                                  CDNO FD  LOC  INSTRUCTION     REF

01  427   *                 LOGIC EXAMPLE 6.
02  428   PROGSW     NOP         REPRESENTS PROGRAM SWITCH                           00001     0325 -0100090000
03  429   *
04  430   ANYLABEL   LOGIC  PROGSW=NOT REGISTERED
05      X ANYLABEL   BES    REGISTERED,M.2                                                     0326 +6100000329
06      X M.1        MSP    PROGSW                                                             0327 -0300910325
07      X            B      M.3                                                                0328 +0100090330
08      X M.2        MSM    PROGSW                                                             0329 -0300610325
09      X M.3        NOP                                                             00002     0330 -0100000000
10      X ORIGIN     CNTRL  *-1
11  431   *
```

LOGIC Example 6

If the electronic switch REGISTERED is ON, the program switch PROGSW will be turned OFF (i.e., its sign will be made minus); if REGISTERED is OFF, PROGSW will be turned ON (plus). The program will continue sequentially in either case.

```
PAGE AA    PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01  435   *                 LOGIC EXAMPLE 7.
02  436   ALTSW      EQU    1,SN
03  437   PROGSW     NOP         REPRESENTS PROGRAM SWITCH                    00001    0325 -0100090000
04  438   TRUEBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH                0326 -0100090000
05  439   FALSEBR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH                0327 -0100090000
06  440   *
07  441   ANYLABEL   LOGIC  PROGSW AND ALTSW,TRUEBR,FALSEBR
08      X ANYLABEL   CSM    PROGSW                                                     0328 -0300600325
09      X            BE     FALSEBR                                                    0329 -4100090327
10      X M.5        BAS    ALTSW,TRUEBR                                  00002    0330 +5100100326
11      X            B      FALSEBR                                                    0331 +0100090327
12  442   *
```

LOGIC Example 7

If both the program switch PROGSW and the alteration switch ALTSW are ON, the program will branch to TRUEBR. If either one or both of these switches are OFF, the program will continue with FALSEBR.

```
PAGE AA    PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01  446   *                 LOGIC EXAMPLE 8.
02  447              DSW    -DIGITSW                                      00001 00 0325 +0000000000     0325
03  448              DA     1                                                           +0003260326
04  449   NET               0,9                                                 09 0326                 0326
05  450   TRUEBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH   00002    0327 -0100090000
06  451   *
07  452   ANYLABEL   LOGIC  DIGITSW OR (NET G 1200),TRUEBR
08      X ANYLABEL   CD     DIGITSW,0                                                  0328 +0300000325
09      X            BH     TRUEBR                                                     0329 -4000090327
10      X M.5        ZA2    NET(0,9)                                                   0330 +2300090326
11      X            S2     +1200                                                      0331 -2400030336
12      X            BV2    *+2                                           00003    0332 +2100090334
13      X            BZ2    *+3                                                        0333 +2000090336
14      X            BM2    *+2                                                        0334 -2000090336
15      X            B      TRUEBR                                                     0335 +0100090327
16  453   *
17  454   *        THE FOLLOWING IS GENERATED OUT OF LINE
18  455   *
                     LITERALS
19      X                   +1200                                               03 0336 +1200           0336
```

LOGIC Example 8

If the digit switch DIGITSW is ON, or if the numerical field NET contains a number greater than +1200, or both, the program will continue with the instruction labeled TRUEBR. Only if DIGITSW is OFF and the NET field contains a number equal to or less than +1200 will the program continue sequentially.

```
PAGE AA     PROGRAM                                         7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL       OP     OPERAND                                           CDNO FD  LOC  INSTRUCTION      REF

01  459   *                  LOGIC EXAMPLE 9.
02  460              DSW     -DIGITSW                                          00001 00 0325 +0000000000      0325
03  461   PROGSW     NOP          REPRESENTS PROGRAM SWITCH                             0326 -0100090000
04  462   *
05  463   ANYLABEL   LOGIC   DIGITSW=NOT(ELECSW AND PROGSW)
06      X ANYLABEL   BES     ELECSW,*+2                                                 0327 +6100000329
07      X            B       M.1                                                        0328 +0100090331
08      X M.5        CSM     PROGSW                                                     0329 -0300600326
09      X            BH      M.2                                               00002    0330 -4000090333
10      X M.1        ZA1     +1111111111                                                0331 +1300090335
11      X            B       M.4                                                        0332 +0100090334
12      X M.2        ZA1     +0                                                         0333 +1300000336
13      X M.4        ST1     DIGITSW                                                    0334 +1200000325
14  464   *
15  465   *        THE FOLLOWING IS GENERATED OUT OF LINE
16  466   *
                     LITERALS
17      X                    +1111111111                                       00003 09 0335 +1111111111      0335
18      X                    +0                                                      00 0336 +0               0336
```

LOGIC Example 9

If both the electronic switch ELECSW and the program switch PROGSW are ON, digit switch DIGITSW will be turned OFF. If either ELECSW or PROGSW is OFF, or both, DIGITSW will be turned ON. In either case, the program will continue sequentially.

```
PAGE AA    PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP    OPERAND                                                  CDNO FD  LOC  INSTRUCTION      REF

01  470   *                LOGIC EXAMPLE 10.
02  471              DA    1                                                                          +0003250325
03  472   MATRIX           0,9                                                             09  0325                   0325
04  473   FALSEBR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE     00001      0326 -0100090000
05  474   *
06  475   ANYLABEL   LOGIC SWITCH(XWORD+125)=(MATRIX(ROW-1) NOT E 1),,  REMARKS
07  476                    FALSEBR(7-3)                                 MAY BE USED.
08          X ANYLABEL ZA2 MATRIX(0,9)-1+ROW                                                   0327 +2301090324
09          X          S2  +1                                                                  0328 -2400000335
10          X          BV2 *+2                                                                 0329 +2100090331
11          X          BZ2 *+2                                                                 0330 +2000090332
12          X          B   M.1                                                      00002      0331 +0100090334
13          X M.2      ESF SWITCH+125+XWORD                                                    0332 +6102200125
14          X          B   FALSEBR-3+X7                                                        0333 +0107090323
15          X M.1      ESN SWITCH+125+XWORD                                                    0334 +6102100125
16  477   *
17  478   *         THE FOLLOWING IS GENERATED OUT OF LINE
18  479   *
                       LITERALS
19          X              +1                                                          00  0335 +1                    0335
```

LOGIC Example 10

If the field defined as MATRIX, indexed by the index word ROW and address-adjusted by -1, is not equal to +1, the electronic switch addressed by SWITCH, indexed by XWORD and address-adjusted by +125, will be turned ON. Otherwise the switch will be turned OFF and the program will continue with the instruction located at FALSEBR incremented by the contents of index word 7 and decremented by 3.

```
PAGE AA    PROGRAM                                         7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA
LN CDREF  LABEL      OP    OPERAND                                          CDNO FD  LOC  INSTRUCTION    REF
01  483   *                      LOGIC EXAMPLE 11
02  484              DA    8,,0+X87
03  485   STATE      CODE  0                                                              +0003250332
04  486    OHIO            5                                                     00  0325                 0000
05  487    NEWYORK         6
06  488    MAINE           7
07  489   TRUEBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE  00001   0333 -0100090000
08  490   FALSEBR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE          0334 -0100090000
09  491   *
10  492   ANYLABEL   LOGIC SWITCH=OHIO,TRUEBR,FALSEBR
11      X ANYLABEL   CD    STATE,5
12      X            BE    M.1                                                       0335 +0387500000
13      X M.2        ESF   SWITCH                                                    0336 -4100090339
14      X            B     FALSEBR                                                   0337 +6100200000
15      X M.1        ESN   SWITCH                                           00002    0338 +0100090334
16      X            B     TRUEBR                                                    0339 +6100100000
17  493   *                                                                          0340 +0100090333
18  494   *        THE FOLLOWING IS GENERATED OUT OF LINE
19  495   *
                     LITERALS
20      X                  5
21      X                  6                                                     00  0341 +5          0341
22      X                  7                                                     11  0341 + 6         0341
                                                                                 22  0341 +  7        0341
```

LOGIC Example 11

This example is similar to Example 5. Additional CODE values have been established corresponding to NEWYORK and OHIO under a DA in which both relative addressing and implicit indexing are used.

ZSIGN—Branch on Test for Zero and Sign

ZSIGN generates instructions that will analyze a field or area for the presence of zeros and, if it is not zero, for its sign, and then branch accordingly.

Source Program Format

The two basic formats for the ZSIGN statement in the source program are as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50 55 (Basic Autocoder)
01	ANYLABEL	ZSIGN	FIELD,ZEROBR,PLUSBR,MINUSBR,ALPHABR
02	ANYLABEL	ZSIGN	FIELD,NOZERO,PLUSBR,MINUSBR,ALPHABR
03			

In these examples, ANYLABEL represents any symbolic label; it may be omitted. The entries ZSIGN and NOZERO must be written exactly as shown. FIELD is the symbolic name of the subsequent entry defining the field to be tested. It may also be the header label of a declarative statement in which case the record area defined will be tested. ZEROBR, PLUSBR, MINUSBR, and ALPHABR are the symbolic labels of instructions to which the program will branch if the contents of the field or area are found to be, respectively, zero, plus, minus, or alpha. In the second format, the entry NOZERO instead of ZEROBR will prevent testing for a zero condition; branching then is purely according to sign.

The source program formats may be modified by the omission of one, two, or in the case of the first format, even three branches. In that case, the object program would take the next instruction following the ZSIGN statement instead of the missing branch. Separating commas must be entered if any branch except the last is omitted.

In the first format, if the field contains zeros, transfer will be to ZEROBR whether the zeros are plus or minus. Alpha blanks (of the form @0000000000) will also cause a branch to ZEROBR. Alpha zeros (@9090909090), on the other hand, will cause the program to transfer to ALPHABR (except for the marginal case in which the field has only one digit and that contains the zero digit of an alpha zero, in which case transfer is to ZEROBR.) In this format, if the field does not contain zeros, transfer will be according to the sign of the word in which the left-most digit of the field is contained.

In the second format, branching will be determined by the sign of the word in which the left-most digit of the field is contained.

In the following examples, assume that the field consists of the underlined digits:

Contents of Storage	*Branch Taken Using the First Format*	*Branch Taken Using the Second Format*
+000<u>1234567</u>	PLUSBR	PLUSBR
−<u>1234567890</u>	MINUSBR	MINUSBR
@<u>8361776979</u>	ALPHABR	ALPHABR

Contents of Storage	*Branch Taken Using the First Format*	*Branch Taken Using the Second Format*
+0000000000	ZEROBR	PLUSBR
−0000000000 +0000000005	ZEROBR	MINUSBR
@0000000000	ZEROBR	ALPHABR
@9090909090	ALPHABR	ALPHABR
@9090909090	ZEROBR	ALPHABR
+0001234567 −1234567890	PLUSBR	PLUSBR
@8272656274 +0000000000	ALPHABR	ALPHABR

When a ZSIGN statement references the label of a DA header line, coding will be generated to cause the following:

1. If the DA header line does not specify a relative address and implicit indexing, the *first* record area defined will be tested as specified.
2. If the DA header line specifies a relative address and implicit indexing, the *current* record area (as determined by the contents of the implicit index word) will be tested as specified.

When a ZSIGN statement references any other declarative statement header line, the entire area will be tested as specified.

Processing Techniques

Limitations on Length

The number of parameters is fixed by the format, subject only to the omission of one or more branches. There is no limit to the size of the field to be tested.

Address Modification

Indexing and address adjustment are permitted on all symbolic addresses.

Error and Warning Messages

The following error and warning messages are issued during assembly under the conditions indicated:

BRANCH TO NON-IMPERATIVE INSTRUCTION

One of the branch addresses to which the object program may transfer is not the label of an imperative instruction, but, for example, that of a DA subsequent entry.

FIELD MISSING

This message will be issued if FIELD is omitted from the operand, e.g., if the operand portion of the source statement begins with a comma.

FIELD UNACCEPTABLE

The FIELD entry contains a symbolic address of an entity that cannot meaningfully be tested for zero contents or sign, e.g., an alteration switch.

NO BRANCHES GIVEN

All four branches in the first format, or all three branches in the second format, have been omitted. A NOP will be generated to aid in patching if this condition was unintended. It should be noted, however, that this instruction will accomplish exactly what a literal interpretation of the source statement requires, i.e., the program will take the next instruction in any case.

UNLIKELY — ALL BRANCHES IDENTICAL

All four branches in the first format, or all three branches in the second format, are identical. In case the source statement is of the first format, coding for the zero test will have been generated. Then the procedure is the same as for the second format, i.e., a NOP will be generated, followed by an unconditional Branch instruction to the required branch This again allows for patching while implementing a strict interpretation of the source statement.

Examples

The following are examples of acceptable coding for the ZSIGN macro-instruction. For each, the associated source-program entries are given, followed by the ZSIGN statement, and coding generated in-line.

```
PAGE AA     PROGRAM                                         7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                                   CDNO FD  LOC  INSTRUCTION     REF

01  501   *                 ZSIGN EXAMPLE 1
02  502              DA     1                                                                       +0003250325
03  503   ANYFIELD          3,9A4.3                                                          39  0325                0325
04  504   ZEROBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE     00001        0326 -0100090000
05  505   PLUSBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                  0327 -0100090000
06  506   MINUSBR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                  0328 -0100090000
07  507   ALPHABR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                  0329 -0100090000
08  508   *
09  509   ANYLABEL   ZSIGN  ANYFIELD,ZEROBR,PLUSBR,MINUSBR,ALPHABR
10      X ANYLABEL   ZA1    ANYFIELD(0,6)                                                    0330 +1300390325
11      X            BZ1    ZEROBR                                              00002        0331 +1000090326
12      X            CSM    ANYFIELD                                                         0332 -0300600325
13      X            BL     ALPHABR                                                          0333 +4000090329
14      X            BL     MINUSBR                                                          0334 -4100090328
15      X            B      PLUSBR                                                           0335 +0100090327
16  510   *
```

ZSIGN Example 1

A field, less than one word in length, will be examined first for the presence of zeros and, if it does not contain all zeros, for a plus, minus, or alpha sign. A branch to the appropriate instruction will be made based on what is found.

```
PAGE AA     PROGRAM                                         7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                                   CDNO FD  LOC  INSTRUCTION     REF

01  514   *                 ZSIGN EXAMPLE 2
02  515              DA     1                                                                       +0003250325
03  516   FIELDX            0,9A                                                             09  0325                0325
04  517   PLUSBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE     00001        0326 -0100090000
05  518   MINUSBR    NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                  0327 -0100090000
06  519   *
07  520   ANYLABEL   ZSIGN  FIELDX,,PLUSBR,MINUSBR
08      X ANYLABEL   ZA1    FIELDX(0,9)                                                      0328 +1300090325
09      X            BZ1    M.1                                                              0329 +1000090333
10      X            CSM    FIELDX                                                           0330 -0300600325
11      X            BH     PLUSBR                                              00002        0331 -4000090326
12      X            BE     MINUSBR                                                          0332 -4100090327
13      X M.1        NOP                                                                     0333 -0100000000
14      X ORIGIN     CNTRL  *-1
15  521   *
```

ZSIGN Example 2

The program will examine the one-word field, FIELDX, for a plus or minus sign and branch accordingly. If FIELDX only contains zeros or has an alpha sign, the program will continue with the next sequential instruction.

```
PAGE AA    PROGRAM                                     7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF LABEL      OP    OPERAND                                                     CDNO FD  LOC  INSTRUCTION     REF

01  525   *               ZSIGN EXAMPLE 3
02  526             DA    1                                                                          +0003250331
03  527   AFIELD          03,69                                                             39  0325                0325
04  528   BRANCHX   NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE          00001     0332 -0100090000
05  529   BRANCHZ   NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                    0333 -0100090000
06  530   *
07  531   ANYLABEL  ZSIGN AFIELD,BRANCHX,BRANCHZ,BRANCHZ,BRANCHX
08      X ANYLABEL  XL    MACREG.01,+0000000005                                                 0334 +4500010343
09      X           ZAA   AFIELD                                                                0335 +1600390325
10      X M.2       AA    AFIELD(7,16)+MACREG.01                                                0336 +1701090326
11      X           BIX   MACREG.01,M.2                                           00002         0337 +4900010336
12      X           BV1   *+2                                                                   0338 +1100090340
13      X           BZ1   BRANCHX                                                               0339 +1000090332
14      X           CSM   AFIELD                                                                0340 -0300600325
15      X           BL    BRANCHX                                                               0341 +4000090332
16      X           B     BRANCHZ                                                 00003         0342 +0100090333
17  532   *
18  533   *        THE FOLLOWING IS GENERATED OUT OF LINE
19  534   *
                    LITERALS
20      X                 +0000000005                                                       09  0343 +0000000005    0343
```

ZSIGN Example 3

The program will examine a field greater than one word. If AFIELD contains zeros or if the sign of the word in which the left-most digit of AFIELD is contained is alpha, a branch to BRANCHX will be made. If the sign is plus or minus, the branch will be to BRANCHZ.

```
PAGE AA    PROGRAM                                     7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF LABEL      OP    OPERAND                                                     CDNO FD  LOC  INSTRUCTION     REF

01  536   *               ZSIGN EXAMPLE 4
02  537             DA    1                                                                          +0003250331
03  538   AFIELD          03,69                                                             39  0325                0325
04  539   NOZERO    NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE          00001     0332 -0100090000
05  540   BRANCHZ   NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                    0333 -0100090000
06  541   BRANCHX   NOP       REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                    0334 -0100090000
07  542   *
08  543   ANYLABEL  ZSIGN AFIELD,NOZERO,BRANCHZ,BRANCHZ,BRANCHX
09      X ANYLABEL  CSM   AFIELD                                                                0335 -0300600325
10      X           BL    BRANCHX                                                               0336 +4000090334
11      X           B     BRANCHZ                                                 00002         0337 +0100090333
12  544   *
```

ZSIGN Example 4

A field greater than one word will be examined for sign only. The program will continue with the instruction located at BRANCHZ if the sign of the word which contains the left-most digit of AFIELD is plus or minus, or BRANCHX if the sign is alpha.

```
PAGE AA     PROGRAM                                                         7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                              CDNO FD  LOC   INSTRUCTION    REF

01  548   *                       ZSIGN EXAMPLE 5
02  549               DA     4,RDW,0+X13                                                       +0003250332
03          X                                                                    00001     0325 +0003290329    0325
04          X                                                                              0326 +0003300330    0326
05          X                                                                              0327 +0003310331    0327
06          X                                                                              0328 -0003320332    0328
07  550   ANYFIELD           3,9A4.3                                                   39  0329                0000
08  551   ZEROBR      NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE 00002     0333 -0100090000
09  552   PLUSBR      NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE           0334 -0100090000
10  553   MINUSBR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE           0335 -0100090000
11  554   ALPHABR     NOP         REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE           0336 -0100090000
12  555   *
13  556   ANYLABEL    ZSIGN  ANYFIELD,ZEROBR,PLUSBR,MINUSBR,ALPHABR
14          X ANYLABEL    ZA1    ANYFIELD(0,6)+X13                                             0337 +1313390000
15          X             BZ1    ZEROBR                                              00003     0338 +1000090333
16          X             CSM    ANYFIELD+X13                                                  0339 -0313600000
17          X             BL     ALPHABR                                                       0340 +4000090336
18          X             BE     MINUSBR                                                       0341 -4100090335
19          X             B      PLUSBR                                                        0342 +0100090334
20  557   *
```

ZSIGN Example 5

The actual ZSIGN test is the same as in Example 1. In this case, however, four blocked records are in the defined area. The test will be made only on the field ANYFIELD in the <u>current</u> record area, as determined by the contents of the implicit index word.

```
PAGE AA    PROGRAM                                     7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP    OPERAND                                                   CDNO  FD  LOC  INSTRUCTION    REF

01  561   *                      ZSIGN EXAMPLE 6
02  562   ANYLABEL    DA    3,,0+INDEXWORD                                                               +0003250333
03  563   FIELDA            0,9A6.4                                                         09  0325                  0000
04  564   FIELDB            10,15'                                                          05  0326                  0001
05  565   FIELDC            16,28                                                           69  0326                  0001
06  566   ZEROBR      NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE    00001       0334 -0100090000
07  567   PLUSBR      NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                0335 -0100090000
08  568   MINUSBR     NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                0336 -0100090000
09  569   ALPHABR     NOP        REPRESENTS FIRST INSTRUCTION OF BRANCH ROUTINE                0337 -0100090000
10  570   *
11  571   SOMELABEL   ZSIGN ANYLABEL,ZEROBR,PLUSBR,          REMARKS MAY
12  572                     MINUSBR,ALPHABR                  BE USED.
13      X SOMELABEL   ZAA   0(0,9)+INDEXWORD                                                   0338 +1601090000
14      X             AA    0(10,19)+INDEXWORD                                     00002       0339 +1701090001
15      X             AA    0(20,29)+INDEXWORD                                                 0340 +1701090002
16      X             BV1   *+2                                                                0341 +1100090343
17      X             BZ1   ZEROBR                                                             0342 +1000090334
18      X             CSM   0+INDEXWORD                                                        0343 -0301600000
19      X             BL    ALPHABR                                                00003       0344 +4000090337
20      X             BE    MINUSBR                                                            0345 -4100090336
21      X             B     PLUSBR                                                             0346 +0100090335
22  573   *
```

ZSIGN Example 6

Since the label of the DA header line is addressed in the ZSIGN statement, the entire area through digit position 29 will be tested for zeros. Since implicit indexing has been used, the current record will be tested. If the record does not contain all zeros, the sign of the word which contains the left-most digit of ANYLABEL (FIELDA) will be checked, and appropriate branches will be made.

SETSW — Set Switch

SETSW generates instructions to set one or more digit, electronic, or program switches to an on or off condition.

Source Program Format

The basic format for the SETSW statement in the source program is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	SETSW	±SWA, ±SWB, ±SWC, etc.
02			

In this example, ANYLABEL is any symbolic label; it may be omitted. The entry SETSW must be written exactly as shown. The entries SWA, SWB, etc., represent names of switches that are to be turned ON or OFF. If the switch is a program or a digit switch, the name must be symbolic; the label of a series of digit switches established by a DSW (Define Switch) statement may be used. If the switch is electronic, either its symbolic or its actual one- or two-digit name may be used. The various types may be freely intermingled within the operand portion of a single SETSW statement.

Switches preceded by a plus sign will be turned ON; those preceded by a minus sign will be turned OFF. The sign may be omitted, in which case it will be assumed to be plus. If a series of switches established by a DSW instruction is included, all of the component switches will be turned ON or OFF according to the sign (or its absence) preceding the label.

Commas must be entered between successive entries in the operand portion of the SETSW statement.

Processing Techniques

Limitations on Length

The operand portion may include 94 parameters. Any excess will be ignored.

Address Modification

All symbolic addresses may be modified by indexing and address adjustment.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

INSTRUCTION NOT PROGRAM SWITCH PAR. *xx*

An attempt has been made to turn a program switch ON or OFF (to make plus or minus an instruction with operation code 01) and the instruction addressed fails to have this operation code. *xx* will be replaced by the number of the parameter at fault. Coding will be generated to implement the sign adjustment nevertheless.

INVALID SWITCH. PARAMETER *xx*

An attempt has been made to set hardware switches, such as alteration switches, by use of the SETSW statement, or other unacceptable operand entries have been made. *xx* will be replaced by the number of the parameter at fault.

Example

The following is an example of acceptable coding for the SETSW macro-instruction. The associated source-program entries are given, followed by the SETSW statement, the coding generated in-line, and the coding out-of-line.

```
PAGE AA     PROGRAM                                   7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN  CDREF  LABEL      OP     OPERAND                                          CDNO  FD  LOC  INSTRUCTION   REF

01  577    *                 SETSW EXAMPLE 1
02  578               DSW    -DIGSWB,+DIGSWA                                  00001 01  0325 +0100000000   0325
03  579    PROGSWA    NOP          REPRESENTS PROGRAM SWITCH                            0326 -0100090000
04  580    PROGSWB    NOP          REPRESENTS PROGRAM SWITCH                            0327 -0100090000
05  581    *
06  582    ANYLABEL   SETSW  ELECSWA,-21,PROGSWA,+DIGSWA,    REMARKS MAY
07  583                      -PROGSWB,-DIGSWB                BE USED.
08       X ANYLABEL   ESN    ELECSWA                                                    0328 +6100100000
09       X            ESF    21                                                         0329 +6300200000
10       X            MSP    PROGSWA                                          00002     0330 -0300910326
11       X            ZA1    +1111111111                                                0331 +1300090336
12       X            ST1    DIGSWA                                                     0332 +1200110325
13       X            MSM    PROGSWB                                                    0333 -0300610327
14       X            ZA2    +0                                                         0334 +2300000337
15       X            ST2    DIGSWB                                           00003     0335 +2200000325
16  584    *
17  585    *        THE FOLLOWING IS GENERATED OUT OF LINE
18  586    *
                      LITERALS
19       X                   +1111111111                                            09  0336 +1111111111   0336
20       X                   +0                                                     00  0337 +0            0337
```

SETSW Example 1

Electronic switch A, program switch A, and digit switch(es) A will be turned ON. Electronic switch 21, program switch B, and digit switch(es) B will be turned OFF.

ZERO — Zero Storage

ZERO generates instructions to replace the contents of fields or areas with zeros or blanks.

Source Program Format

The basic format for the zero statement in the source program is as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
01	ANYLABEL	ZERO	FIELDA, AREAB, FIELDC, etc.
02			

ANYLABEL is any symbolic label; it may be omitted. The entry ZERO must be written exactly as shown. FIELDA, AREAB, etc., may be the symbolic names of any defined fields or areas. Areas, numerical fields and alphameric fields may be freely intermingled.

Processing Techniques

Limitations on Length

The operand portion of the ZERO macro-instruction may contain up to 94 entries. There is no limit on the size of the fields named.

Address Modification

All symbolic addresses may be modified by indexing and address adjustment.

The Effect of ZERO

Fields and Areas. DLINE areas and all fields defined as alphameric will be replaced by the double-digit representation of blanks. Whole words will be made alpha; the sign of partial words will not be altered.

In all other fields or areas, whole words will be replaced by plus zeros; partial words will be replaced by ZEROs but the sign will not be altered.

The following examples illustrate the effect of ZERO on various fields. The field addressed is underlined.

No.	Field Definition	Before ZERO	After ZERO
1.	0, 9A10.0	−8342168900	+0000000000
2.	0, 9@	@9192616263	@0000000000
3.	0, 13A	−9342168900 −1869123456	+0000000000 −0000123456
4.	0, 13A	−8342168900 +1869123456	+0000000000 +0000123456
5.	0, 15@	@9192616262 @6162636465	@0000000000 @0000006465
6.	0, 15@	−9192616263 −6162636465	@0000000000 −0000006465
7.	4, 8A5.0	−1234012345	−1234000005
8.	4, 8A5.0	@8283909195	@8283000005

In example 8, machine difficulties may arise when an attempt is made to print out the zeroed word, since the combination 05 has no meaning in double-digit, alphameric code.

Declarative Statement Header Lines. When ZERO references the label of a DA header line, coding will be generated to cause the following:

1. If the DA header line does not specify a relative address and implicit indexing, the *first* record area defined will be set to plus zeros.
2. If the DA header line specifies a relative address and implicit indexing, the *current* record area (as determined by the contents of the implicit index word) will be set to plus zeros.

If ZERO references the label of a DLINE header line, coding will be generated to cause the entire area, including constants (if any), to be set to blanks.

If ZERO references the label of a DRDW, a warning message will be issued, but coding will be generated to cause the first RDW generated (*not* the area it defines) to be set to plus zeros.

When ZERO references the label of any other declarative statement header line, coding will be generated to cause the entire area to be set to plus zeros.

Instructions. If ZERO references the label of an instruction, a warning message will be issued, but coding will be generated to cause the instruction to be set to plus zeros.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

ALPHA BLANKS INTO UNDEFINED PAR. *xx*

The *xx* will be replaced by the parameter number of the operand without defined characteristics. This field will be filled with alphameric blanks.

ATTEMPTING TO ZERO HARDWARE. PAR. *xx*

The field to be zeroed has been defined by means of an EQU line as a hardware device. A NOP will be generated.

NO FIELD SIZE. PAR. *xx*

The parameter record of the operand entry whose number replaces the *xx* of the message does not indicate the size of the field to be zeroed out. A NOP will be generated.

ZEROING DC. PAR. *xx*

The label of a DC header line has been used as an operand. The parameter number will replace the *xx* of the warning message. Coding will be generated nevertheless.

ZEROING INSTRUCTION. PAR. *xx*

This warning message, with the parameter number of the faulty entry in place of the *xx*, will be issued whenever an attempt is made to zero out an instruction, whether symbolic machine or macro. Coding will be generated, however.

Examples

The following are examples of acceptable coding for the ZERO macro-instruction. For each, the associated source-program entries are given, followed by the ZERO statement, coding generated in-line, and coding generated out-of-line.

```
PAGE AA    PROGRAM                                            7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP    OPERAND                                                  CDNO FD  LOC  INSTRUCTION      REF

01  601   *                ZERO EXAMPLE 1
02  602              DA    1                                                                      +0003250438
03  603   FIELDA           00,99'                                                        09  0325                  0325
04  604   FIELDB           100,103A                                                      03  0335                  0335
05  605   FIELDC           110,129A                                                      09  0336                  0336
06  606   FIELDD           132,1131A                                                     29  0338                  0338
07  607   *
08  608   ANYLABEL   ZERO  FIELDA,FIELDB,FIELDC,       REMARKS MAY
09  609                    FIELDD                      BE USED.
10      X ANYLABEL   ZA1   ' '                                                      00001    0439 +1300010455
11      X            ST1   FIELDA(0,9)                                                       0440 +1200090325
12      X            XZA   MACREG.1,FIELDA                                                   0441 +4600010325
13      X            RS    MACREG.1,M.2                                                      0442 +6500010452
14      X            STD1  FIELDB(0,3)                                                       0443 -1200030335
15      X            ZA2   +0                                                       00002    0444 +2300000454
16      X            ST2   FIELDC(0,9)                                                       0445 +2200090336
17      X            ST2   FIELDC(10,19)                                                     0446 +2200090337
18      X            STD1  FIELDD(0,7)                                                       0447 -1200290338
19      X            ST2   FIELDD(8,17)                                                      0448 +2200090339
20      X            XZA   MACREG.1,FIELDD+1                                        00003    0449 +4600010339
21      X            RS    MACREG.1,M.3                                                      0450 +6500010453
22      X            STD1  FIELDD(998,999)                                                   0451 -1200010438
23  610    *
24  611    *       THE FOLLOWING IS GENERATED OUT OF LINE
25  612    *
26      X M.2        DRDW  -FIELDA+1,FIELDA+9                                                0452 -0003260334
27      X M.3        DRDW  -FIELDD+2,FIELDD+99                                               0453 -0003400437
                     LITERALS
28      X                  +0                                                       00004 00 0454 +0           0454
29      X                  ' '                                                      00005 01 0455 '00          0455
```

ZERO Example 1

FIELDA will be filled with alphameric blanks. FIELDB will be filled with zeros; the sign of the word will not be changed. FIELDC will be filled with plus zeros. The first and last words in which digits of FIELDD occur will have the portion occupied by FIELDD replaced by zeros; the sign of these words will not be changed. The rest of FIELDD will be filled with plus zeros.

```
PAGE AA    PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                          CDNO FD  LOC  INSTRUCTION    REF

01  616   *                ZERO EXAMPLE 2
02  617              DA    2,RDW,0+INDEXWORD                                             +0003250352
03      X                                                                   00001    0325 +0003270339    0325
04      X                                                                            0326 -0003400352    0326
05  618   FIELDA           00,95'                                                 09 0327                0000
06  619   FIELDB           96,103A                                                69 0336                0009
07  620   FIELDC           110,129A                                               09 0338                0011
08  621   *
09  622   ANYLABEL   ZERO  FIELDA,FIELDB,FIELDC
10      X ANYLABEL   ZA1   ' '                                              00002    0353 +1300010366
11      X            ST1   FIELDA(0,9)+INDEXWORD                                     0354 +1201090000
12      X            XZS   MACREG.2,FIELDA+1+INDEXWORD                               0355 -4601020001
13      X            XSN   MACREG.2,FIELDA+8+INDEXWORD                               0356 +4801020008
14      X            XZA   MACREG.1,FIELDA+INDEXWORD                                 0357 +4601030000
15      X            RS    MACREG.1,MACREG.2                                00003    0358 +6500030002
16      X            STD1  FIELDA(90,95)+INDEXWORD                                   0359 -1201050009
17      X            STD1  FIELDB(0,3)+INDEXWORD                                     0360 -1201690009
18      X            STD1  FIELDB(4,7)+INDEXWORD                                     0361 -1201030010
19      X            ZA2   +0                                                        0362 +2300000365
20      X            ST2   FIELDC(0,9)+INDEXWORD                            00004    0363 +2201090011
21      X            ST2   FIELDC(10,19)+INDEXWORD                                   0364 +2201090012
22  623   *
23  624   *        THE FOLLOWING IS GENERATED OUT OF LINE
24  625   *
                     LITERALS
25      X                  +0                                                     00 0365 +0         0365
26      X                  ' '                                              00005 01 0366 '00        0366
```

ZERO Example 2

In the current record area, as determined by the contents of the implicit index word, FIELDA will be filled with alphameric blanks and FIELDB will be filled with zeros. Note that the last word that contains digits of FIELDA also contains digits of FIELDB. The sign of this word, therefore, will not be changed. The sign of the last word which contains digits of FIELDB will also remain unchanged since it is only a partial word. The contents of FIELDC will be replaced with plus zeros.

```
PAGE AA     PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION        REF

01  629   *                  ZERO EXAMPLE 3
02  630   AREANAME    DA     1                                                              +0003250349
03  631   FIELDA             00,99'                                               09  0325                     0325
04  632   FIELDB             100,103A                                             03  0335                     0335
05  633   FIELDC             104,129A                                             49  0335                     0335
06  634   FIELDD             130,249A                                             09  0338                     0338
07  635   *
08  636   ANYLABEL    ZERO   AREANAME
09      X ANYLABEL    ZA2    +0                                           00001      0350 +2300000355
10      X             ST2    AREANAME(0,9)                                           0351 +2200090325
11      X             XZA    MACREG.1,AREANAME                                       0352 +4600010325
12      X             RS     MACREG.1,M.2                                            0353 +6500010354
13  637   *
14  638   *        THE FOLLOWING IS GENERATED OUT OF LINE
15  639   *
16      X M.2         DRDW   -AREANAME+1,AREANAME+24                                 0354 -0003260349
                      LITERALS
17      X                    +0                                           00002 00   0355 +0                   0355
```

ZERO Example 3

Since the ZERO statement references the label of the DA header line, the entire record area defined will be filled with plus zeros.

```
PAGE AA     PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION        REF

01  643   *                  ZERO EXAMPLE 4
02  644   AREANAME    DA     2,RDW,0+INDEXWORD
03      X                                                                                   +0003250348
04      X                                                                 00001      0325 +0003270337      0325
                                                                                     0326 -0003380348      0326
05  645   FIELDA             00,99'                                               09  0327                     0000
06  646   FIELDB             100,103A                                             03  0337                     0010
07  647   *
08  648   ANYLABEL    ZERO   AREANAME
09      X ANYLABEL    ZA2    +0                                           00002      0349 +2300000355
10      X             ST2    0(0,9)+INDEXWORD                                        0350 +2201090000
11      X             XZS    MACREG.2,0+1+INDEXWORD                                  0351 -4601020001
12      X             XSN    MACREG.2,0+10+INDEXWORD                                 0352 +4801020010
13      X             XZA    MACREG.1,0+INDEXWORD                                    0353 +4601030000
14      X             RS     MACREG.1,MACREG.2                            00003      0354 +6500030002
15  649   *
16  650   *        THE FOLLOWING IS GENERATED OUT OF LINE
17  651   *
                      LITERALS
18      X                    +0                                                 00   0355 +0                   0355
```

ZERO Example 4

The ZERO statement in this example references the label of a DA header line which specifies a relative address and implicit indexing. The contents of the current record area will be filled with plus zeros.

```
PAGE AA     PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP    OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01  655   *                 ZERO EXAMPLE 5
02  656   DRDWNAME    DRDW  AREANAME
03      X DRDWNAME    DRDW  +AREANAME,AREANAME+24                            00001    0325 +0003270351
04      X             DRDW  -AREANAME+25,AREANAME+49                                  0326 -0003520376
05  657   *
06  658   AREANAME    DA    2,,0+INDEXWORD                                                 +0003270376
07  659   FIELDA            00,99'                                                 09 0327                 0000
08  660   FIELDB            100,103A                                               03 0337                 0010
09  661   FIELDC            104,129A                                               49 0337                 0010
10  662   FIELDD            130,249A                                               09 0340                 0013
11  663   *
12  664   ANYLABEL    ZERO  DRDWNAME                                                                          M
13      X ANYLABEL    ZA2   +0                                               00002    0377 +2300000379
14      X             STD2  DRDWNAME(0,9)                                             0378 -2200090325
15  665   *
16  666   *         THE FOLLOWING IS GENERATED OUT OF LINE
17  667   *
                      LITERALS
18      X                   +0                                                     00 0379 +0              0379
```

```
ERROR MESSAGE LIST

PG/LN    MESSAGE

AA 12 ZEROING INSTRUCTION. PAR.000001000A
```

ZERO Example 5

The warning message shown will be issued since the ZERO statement references the label of a DRDW. Coding has been generated, however, to cause the first RDW generated to be filled with plus zeros.

```
PAGE AA    PROGRAM                                7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                    CDNO  FD  LOC  INSTRUCTION    REF

01  671   *                 ZERO EXAMPLE 6
02  672   LINENAME   DLINE                                             00005          +0003250348
03  673                     1'1'                                       00006 01  0325 '91            0325
04  674                     10'TOTAL'                                        89  0326 '        83    0326
05      X                                                                    07  0327 '76836173      0327
06  675                     18'$'                                            45  0328 '    25        0328
07  676   GROSSAMT          19X,XXX.ZZ)DR,CR                                 69  0328                0328
08  677   CHECKAMT          60$X,XXZ.ZZ),C                                   89  0336                0336
09  678   ITEMNAME          80,94                                      00007 89  0340                0340
13  679   FLVAR             95F                                              89  0343                0343
14  680                     120''R                                     00008 89  0348 '        80    0348
15  681   *
16  682   ANYLABEL   ZERO   CHECKAMT
17      X ANYLABEL   ZA2    +0                                         00009     0349 +2300000354
18      X            STD2   CHECKAMT(0,1)                                        0350 -2200890336
19      X            ZA1    ' '                                                  0351 +1300010355
20      X            ST1    CHECKAMT(2,11)                                       0352 +1200090337
21      X            STD1   CHECKAMT(12,19)                                      0353 -1200070338
22  683   *
23  684   *        THE FOLLOWING IS GENERATED OUT OF LINE
24  685   *
                     LITERALS
25      X                   +0                                         00010 00  0354 +0             0354
26      X                   ' '                                        00011 01  0355 '00            0355
```

ZERO Example 6

The contents of the DLINE field CHECKAMT is replaced with alphameric blanks. The sign of whole words in the field are set to alpha; the sign of partial words are not altered.

```
PAGE AA    PROGRAM                                     7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL       OP    OPERAND                                        CDNO FD  LOC  INSTRUCTION      REF

01  689   *                 ZERO EXAMPLE 7
02  690   LINENAME    DLINE                                                00002         +0003250333
03  691   AMOUNTONE         19$X,XXX.ZZ                                       69   0328                    0328
04  692   AMOUNTTWO         30$X,XXX,XXX.XX                                   89   0330                    0330
05  693   *
06  694   ANYLABEL    ZERO  LINENAME
07      X ANYLABEL    ZA1   ' '                                            00003    0334 +1300010339
08      X             ST1   LINENAME(0,9)                                           0335 +1200090325
09      X             XZA   MACREG.1,LINENAME                                       0336 +4600010325
10      X             RS    MACREG.1,M.2                                            0337 +6500010338
11  695   *
12  696   *         THE FOLLOWING IS GENERATED OUT OF LINE
13  697   *
14      X M.2         DRDW  -LINENAME+1,LINENAME+8                                  0338 -0003260333
                      LITERALS
15      X                   ' '                                            00004 01 0339 '00              0339
```

ZERO Example 7

The entire area, including constants, will be filled with alphameric blanks since the label of the DLINE header line is referenced in the ZERO statement.

```
PAGE AA    PROGRAM                                     7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL       OP    OPERAND                                        CDNO FD  LOC  INSTRUCTION      REF

01 6901   *               ZERO EXAMPLE 8
02 6902   AREANAME    DA    1                                                            +0003250331
03 6903   FIELDA            00,23'                                            09   0325                    0325
04 6904   FIELDB            30,35A                                            05   0328                    0328
05 6905   FIELDC            45,55                                             59   0329                    0329
06 6906                     65                                                55   0331                    0331
07 6907   ANYLABEL    ZERO  AREANAME
08      X ANYLABEL    ZA2   +0                                             00001    0332 +2300000337
09      X             ST2   AREANAME(0,9)                                           0333 +2200090325
10      X             XZA   MACREG.1,AREANAME                                       0334 +4600010325
11      X             RS    MACREG.1,M.2                                            0335 +6500010336
12 6908   *
13 6909   *         THE FOLLOWING IS GENERATED OUT OF LINE
14 6910   *
15      X M.2         DRDW  -AREANAME+1,AREANAME+6                                  0336 -0003260331
                      LITERALS
16      X                   +0                                             00002 00 0337 +0               0337
```

ZERO Example 8

Since the label of the DA header line is referenced by the ZERO statement, the entire area through digit position 69 will be filled with plus zeros.

FILL — Fill Storage

FILL generates instructions to replace the contents of fields or areas with a specified constant.

Source Program Format

The basic format for the FILL statement in the source program is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–60) Basic Autocoder	
01	ANYLABEL	FILL	FIELDA, AREA1 WITH ±h AND FIELDB WITH @Z@	etc.
02				

ANYLABEL is any symbolic label; it may be omitted. The entry FILL and the AND and WITH separators must be written exactly as shown. FIELDA, AREA1, etc., may be the symbolic names of any defined fields or areas. Areas, numerical fields, and alphameric fields may be freely intermingled.

Processing Techniques

Limitations on Length

The operand portion of the FILL macro-instruction may contain up to 94 entries, including areas and fields to be filled, the WITH and AND separators, and the characters to be inserted. There is no limit on the size of the fields named.

Address Modification

All symbolic addresses may be modified by indexing and address adjustment.

The Effect of FILL

Fields and Areas. The sign of the words in the field or area to be filled will not affect the generated instructions. Each word in the entire field or area, and any word in which a segment of the field may appear, will be set to the sign of the filling constant. It is thus possible to introduce invalid alpha combinations in the following two cases:

1. The field specified is numerical and occupies part of a word(s) and is being filled with an alphameric character.
2. A field with an odd number of digits, or a field that begins in an odd-numbered position, is being filled with an alphameric character.

In case 2, a warning message will be issued.

If two fields are specified in the same word, and each field is to be filled with a different constant, the sign of the word is determined by the sign of the constant which fills a field last.

The following examples illustrate the effect of FILL on various fields. The field addressed is underlined.

No.	Field Definition	Before FILL	Filling Constant	After FILL
1.	00,09A10.0	−1234567890	+9	+9999999999
2.	00,05A4.2	−1234560000	+1	+1111110000
3.	05,14	+6789523721 +4376411111	@Y@	@6789588888@8888811111
4.	08,08	+1234567890	−1	−1234567190
5.	00,15@	@0061626364 @6564630000	+9	+9999999999 +9999990000
6.	00,03	+1234567890	−1	−1111567890
	04,09	−1111567890	+2	+1111222222

In example 3, invalid, double-digit combinations (58, 81, and 11) are introduced. These may cause machine difficulties when an attempt is made to print out these words.

Declarative Statement Header Lines. When FILL references the label of a DA header line, coding will be generated to cause the following:

1. If the DA header line does not specify a relative address and implicit indexing, the *first* record area defined will be filled with the specified value.
2. If the DA header line specifies a relative address and implicit indexing, the *current* record area (as determined by the contents of the implicit index word) will be filled with the specified value.

If FILL references the label of a DLINE header line, coding will be generated to cause the entire area, including constants (if any), to be filled.

If FILL references the label of a DRDW, a warning message will be issued, but coding will be generated to cause the first RDW generated (*not* the area it defines) to be filled.

When FILL references the label of any other declarative statement header line, coding will be generated to cause the entire area to be filled.

Instructions. If FILL references the label of an instruction, a warning message will be issued, but coding will be generated to cause the instruction to be filled.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

ATTEMPTING TO FILL HARDWARE. PAR. *xx*

The field to be filled has been defined by means of an EQU line as a hardware device. A NOP will be generated.

FILLING INSTRUCTION. PAR. *xx*

This warning message, with the parameter number of the faulty entry in place of the *xx*, will be issued whenever an attempt is made to fill an instruction, whether symbolic machine or macro. Coding will be generated, however.

NO FIELD SIZE. PAR. *xx*

The parameter record of the operand entry, whose number replaces the *xx* of the message, does not indicate the size of the field to be filled. A NOP will be generated.

WARNING. INVALID ALPHA MAY BE INTRODUCED.

An alphameric character is filling a field with an odd number of digits, or a field that begins in an odd-numbered position.

Examples

The following are examples of acceptable coding for the FILL macro-instruction. For each, the associated source-program entries are given, followed by the FILL statement, coding generated in-line, and coding generated out-of-line.

```
PAGE AA    PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL     OP    OPERAND                                                CDNO FD  LOC  INSTRUCTION     REF

01  902   *            FILL EXAMPLE 1
02  903             DA     1                                                                    +0003250328
03  904   FIELDA          06,09A                                                      69  0325                    0325
04  905   FIELDB          10,19A                                                      09  0326                    0326
05  906   FIELDC          28,31A                                                      89  0327                    0327
06  907   *
07  908   ANYLABEL  FILL  FIELDA WITH +1 AND FIELDB WITH -1 AND FIELDC WITH 'Y'                                       M
08      X ANYLABEL  ZA1   +1111111111                                        00001        0329 +1300090336
09      X           ST1   FIELDA(0,3)                                                     0330 +1200690325
10      X           ZA1   -1111111111                                                     0331 +1300090337
11      X           ST1   FIELDB(0,9)                                                     0332 +1200090326
12      X           ZA1   'YYYYY'                                                         0333 +1300090338
13      X           ST1   FIELDC(0,1)                                        00002        0334 +1200890327
14      X           ST1   FIELDC(2,3)                                                     0335 +1200010328
15  909   *
16  910   *       THE FOLLOWING IS GENERATED OUT OF LINE
17  911   *
                    LITERALS
18      X                 +1111111111                                            09  0336 +1111111111     0336
19      X                 -1111111111                                            09  0337 -1111111111     0337
20      X                 'YYYYY'                                          00003 09  0338 '8888888888     0338
```

```
ERROR MESSAGE LIST

PG/LN    MESSAGE

AA 07 WARNING-INVALID ALPHA MAY BE INTRODUCED
```

FILL Example 1

FIELDA and FIELDB will be filled with 1s and the sign of the words in which each appears will be set to plus and minus respectively. FIELDC will be filled with Ys and, since it bridges words, the signs of the two words in which it appears will be set to alpha.

```
PAGE AA    PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                                  CDNO FD  LOC  INSTRUCTION    REF

01  914   *          FILL EXAMPLE 2
02  915              DA    2                                                                      +0003250332
03  916   AFIELD           00,09'                                                        09  0325                 0325
04  917   BFIELD           10,15'                                                        05  0326                 0326
05  918   CFIELD           20,23'                                                        03  0327                 0327
06  919   DFIELD           30,33'                                                        03  0328                 0328
07  920   *
08  921   ANYLABEL   FILL  AFIELD WITH ' ' AND BFIELD,CFIELD WITH +1 AND DFIELD
09  922                    WITH -1
10      X ANYLABEL   ZA1   '     '                                                  00001    0333 +1300090342
11      X            ST1   AFIELD(0,9)                                                       0334 +1200090325
12      X            ZA1   +1111111111                                                       0335 +1300090340
13      X            ST1   BFIELD(0,5)                                                       0336 +1200050326
14      X            ST1   CFIELD(0,3)                                                       0337 +1200030327
15      X            ZA1   -1111111111                                              00002    0338 +1300090341
16      X            ST1   DFIELD(0,3)                                                       0339 +1200030328
17  923   *
18  924   *        THE FOLLOWING IS GENERATED OUT OF LINE
19  925   *
                     LITERALS
20      X                  +1111111111                                                   09  0340 +1111111111     0340
21      X                  -1111111111                                                   09  0341 -1111111111     0341
22      X                  '     '                                                  00003 09  0342 '0000000000     0342
```

FILL Example 2

In the <u>first</u> record area defined, AFIELD will be filled with alphameric blanks. BFIELD and CFIELD will be filled with 1s and the sign of the word in which each appears will be set to plus. DFIELD will also be filled with 1s, but the sign of the word will be set to minus.

```
PAGE AA     PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.    PAGE AA

LN CDREF  LABEL        OP     OPERAND                                          CDNO FD   LOC  INSTRUCTION      REF

01   928   *                  FILL EXAMPLE 3                                                  +0003250332
02   929                DA    2,RDW,0+INDEXWORD                                00001      0325 +0003270329      0325
03          X                                                                             0326 -0003300332      0326
04          X                                                                       01    0327                  0000
05   930    A                 00,01                                                 01    0328                  0001
06   931    B                 10,11                                                 01    0329                  0002
07   932    C                 20,21
08   933    *
09   934    ANYLABEL    FILL  A WITH +9 AND B WITH -9 AND C WITH 'Z'           00002      0333 +1300090339
10          X ANYLABEL  ZA1   +9999999999                                                 0334 +1201010000
11          X           ST1   A(0,1)+INDEXWORD                                            0335 +1300090340
12          X           ZA1   -9999999999                                                 0336 +1201010001
13          X           ST1   B(0,1)+INDEXWORD                                            0337 +1300090341
14          X           ZA1   'ZZZZZ'                                          00003      0338 +1201010002
15          X           ST1   C(0,1)+INDEXWORD
16   935    *
17   936    *         THE FOLLOWING IS GENERATED OUT OF LINE
18   937    *
                        LITERALS
19          X                 +9999999999                                           09    0339 +9999999999      0339
20          X                 -9999999999                                           09    0340 -9999999999      0340
21          X                 'ZZZZZ'                                          00004 09   0341 '8989898989      0341
```

FILL Example 3

In the current record area, as determined by the contents of the implicit index word, fields A and B will be filled with 9s and the sign of the word in which each appears will be set to plus and minus respectively. Field C will be filled with Zs and the sign of the word will be set to alpha.

```
PAGE AA     PROGRAM                                7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                   CDNO FD  LOC  INSTRUCTION     REF

01  940   *               FILL EXAMPLE 4
02  941   AREANAME   DA    1                                        +0003250327
03  942   FIELDA           00,03A                          03  0325                    0325
04  943   FIELDB           10,13'                          03  0326                    0326
05  944   FIELDC           20,23                           03  0327                    0327
06  945   *
07  946   ANYLABEL   FILL  AREANAME WITH +0
08      X ANYLABEL   ZA1   +0000000000               00001     0328 +1300090332
09      X            ST1   AREANAME(0,9)                       0329 +1200090325
10      X            ST1   AREANAME(10,19)                     0330 +1200090326
11      X            ST1   AREANAME(20,29)                     0331 +1200090327
12  947   *
13  948   *     THE FOLLOWING IS GENERATED OUT OF LINE
14  949   *
                     LITERALS
15      X                  +0000000000                     09  0332 +0000000000    0332
```

FILL Example 4

Since the FILL statement references the label of the DA header line, the entire record area through digit position 29 will be filled with 0s and the signs of the three words affected will be set to plus.

```
PAGE AA     PROGRAM                                7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                   CDNO FD  LOC  INSTRUCTION     REF

01  952   *               FILL EXAMPLE 5
02  953   AREANAME   DA    2,RDW,0+INDEXWORD                        +0003250332
03      X                                            00001     0325 +0003270329    0325
04      X                                                      0326 -0003300332    0326
05  954   FIELDA           00,03A                          03  0327                    0000
06  955   FIELDB           10,13'                          03  0328                    0001
07  956   FIELDC           20,23                           03  0329                    0002
08  957   *
09  958   ANYLABEL   FILL  AREANAME WITH -0
10      X ANYLABEL   ZA1   -0000000000               00002     0333 +1300090337
11      X            ST1   0(0,9)+INDEXWORD                    0334 +1201090000
12      X            ST1   0(10,19)+INDEXWORD                  0335 +1201090001
13      X            ST1   0(20,29)+INDEXWORD                  0336 +1201090002
14  959   *
15  960   *     THE FOLLOWING IS GENERATED OUT OF LINE
16  961   *
                     LITERALS
17      X                  -0000000000                     09  0337 -0000000000    0337
```

FILL Example 5

Since the label of the DA header line is addressed in the FILL statement, the entire area through digit position 29 will be affected. Since implicit indexing has been used, the current record area will be filled with zeros and the signs of the three words affected will be set to minus.

```
PAGE AA    PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                           CDNO FD  LOC  INSTRUCTION    REF

01  964   *              FILL EXAMPLE 6
02  965   AREANAME   DA    2,,0+INDEXWORD                                                  +0003250344
03  966   AFIELD           00,19A                                                09  0325                 0000
04  967   BFIELD           20,49'                                                09  0327                 0002
05  968   CFIELD           50,99                                                 09  0330                 0005
06  969   *
07  970   DRDWNAME   DRDW  AREANAME
08      X DRDWNAME   DRDW  +AREANAME,AREANAME+9                             00001      0345 +0003250334
09      X            DRDW  -AREANAME+10,AREANAME+19                                    0346 -0003350344
10  971   *
11  972   ANYLABEL   FILL  DRDWNAME WITH +0
12      X ANYLABEL   ZA1   +0000000000                                                 0347 +1300090349
13      X            ST1   DRDWNAME(0,9)                                               0348 +1200090345
14  973   *
15  974   *       THE FOLLOWING IS GENERATED OUT OF LINE
16  975   *
                     LITERALS
17      X                  +0000000000                                           09  0349 +0000000000    0349
```

```
ERROR MESSAGE LIST

PG/LN    MESSAGE

AA 11 FILLING INSTRUCTION
```

FILL Example 6

The warning message shown will be issued since the FILL statement references the label of a DRDW. Coding has been generated, however, to cause the first RDW generated to be filled with 0s and the sign set to plus.

```
AGE AA    PROGRAM                                           7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP    OPERAND                                           CDNO FD  LOC  INSTRUCTION    REF

01  978   *              FILL EXAMPLE 7
02  979   LINENAME   DLINE                                                  00002          +0003250334
03  980   GROSSAMT         10$XX,ZZZ.ZZ                                          89  0326                 0326
04  981   TOTEXPENSE       25$XX,XXX.ZZ                                          89  0329                 0329
05  982   NETAMT           40$XX,ZZZ.ZZ                                          89  0332                 0332
06  983   *
07  984   ANYLABEL   FILL  TOTEXPENSE WITH +0
08      X ANYLABEL   ZA1   +0000000000                                      00003      0335 +1300090339
09      X            ST1   TOTEXPENSE(0,1)                                             0336 +1200890329
10      X            ST1   TOTEXPENSE(2,11)                                            0337 +1200090330
11      X            ST1   TOTEXPENSE(12,19)                                           0338 +1200070331
12  985   *
13  986   *       THE FOLLOWING IS GENERATED OUT OF LINE
14  987   *
                     LITERALS
15      X                  +0000000000                                           09  0339 +0000000000    0339
```

FILL Example 7

The DLINE field TOTEXPENSE will be filled with 0s and the signs of the words affected will be set to plus.

```
PAGE AA    PROGRAM                                     7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP    OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01  990   *               FILL EXAMPLE 8
02  991   LINENAME    DLINE                                                  00002             +0003250333
03  992                     1'      '                                        00003 09  0325 '0000000000      0325
04  993   CUSTOMER          10,29                                                  89  0326                  0326
09  994   AMOUNT            30$XXX,ZZZ.ZZ)DR,CR                                    89  0330                  0330
10  995   *
11  996   ANYLABEL    FILL  LINENAME WITH +0
12      X ANYLABEL    ZA1   +0000000000                                      00004     0334 +1300090339
13      X             ST1   LINENAME(0,9)                                              0335 +1200090325
14      X             XZA   MACREG.01,LINENAME                                         0336 +4600010325
15      X             RS    MACREG.01,M.1                                              0337 +6500010338
16  997   *
17  998   *       THE FOLLOWING IS GENERATED OUT OF LINE
18  999   *
19      X M.1         DRDW  -LINENAME+1,LINENAME+8                                     0338 -0003260333
                      LITERALS
20      X                   +0000000000                                      00005 09  0339 +0000000000      0339
```

FILL Example 8

The entire area, including constants, will be filled with 0s and the signs of the words affected will be made plus since the label of the DLINE header line is referenced in the FILL statement.

EDMOV—Edit and Move Data

EDMOV generates instructions to transfer data between specified fields in storage and to edit them to conform to the format of the field to which they are moved.

Source Program Format

The basic format of the EDMOV statement in the source program is the following:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–60) Basic Autocoder →	Autoc (61–)
01	ANYLABEL	EDMOV	FROMFIELD1 TO TOFIELD1,FROMFIELD2 TO TOF	IELD2,,etc.,
02				

ANYLABEL is any symbolic label; it may be omitted. The entry EDMOV must be written exactly as shown.

FROMFIELD entries may be literals or symbolic addresses of areas from which data is to be moved. The data of FROMFIELD may be alphameric, automatic-decimal, floating-decimal, mixed, or of unspecified characteristics. Automatic-decimal and floating-decimal data may be in single- or double-digit representation; data which is mixed or of unspecified characteristics is treated in the same way as alphameric data.

TOFIELD entries are symbolic addresses of areas to which the edited data is to be moved. A TOFIELD area may be any field defined as alphameric, automatic-decimal, or floating-decimal, or may be a print format defined in a DLINE subsequent entry. Automatic-decimal and floating-decimal data may be in a single- or double-digit representation.

The operator TO must be written exactly as shown, preceded and followed by a blank. Each entry must have both a FROMFIELD and a TOFIELD. If more than one entry is made in the operand of a statement, commas must be used to separate them.

Processing Techniques

Limitations on Length

A maximum of four entries may be written in the operand. Field size is restricted as follows:

Field	*Size*
Automatic-decimal, single-digit	not more than 20 digits
Automatic-decimal, double-digit	not more than 20 characters (40 digits)
Floating-decimal, single-digit	exactly one word
Floating-decimal, double-digit	exactly two words
Alphameric, mixed, unspecified	unrestricted

Other Limitations

DA, DC, and DLINE header lines cannot be used as parameters in an EDMOV statement.

Address Modification

All symbolic addresses may be modified by indexing and address adjustment.

The Effect of EDMOV

All alphameric, mixed, and unspecified fields will be passed on to the MOVE macro generator for processing. No editing is done.

Each of the twenty-two possible types of numerical editing is accomplished by a sequence of steps selected from the following nine basic types of conversion:

1. Automatic-decimal, single-digit	to Automatic-decimal, double-digit
2. Floating-decimal, single-digit	to Floating-decimal, double-digit
3. Automatic-decimal, double-digit	to Automatic-decimal, single-digit
4. Floating-decimal, double-digit	to Floating-decimal, single-digit
5. Automatic-decimal, old format	to Automatic-decimal, new format
6. Automatic-decimal	to Floating-decimal
7. Floating-decimal	to Automatic-decimal
8. Automatic-decimal, double-digit	to Print format
9. Floating-decimal, double-digit	to Print format

The chart on page 207 shows the sequence for each of the twenty-two types of editing.

The rules governing each of the nine types of conversion are as follows:

1. *Automatic-Decimal, Single-Digit to Automatic-Decimal, Double-Digit.* Conversion is accompanied by sign conrol; positive numbers will show a 6 in the next-to-last digit, negative numbers a 7. The sign position of the converted number will have an @. If the result is stored in part of a word, the sign of the entire word will be set to @. The following examples illustrate this conversion:

Before Conversion	*After Conversion*
+7627	@bb97969267
−4502	@bb94959072

2. *Floating-Decimal, Single-Digit to Floating-Decimal, Double-Digit.* FROMFIELD must occupy exactly one word; TOFIELD will occupy exactly two words, both of which will have an @ sign. Sign control will be indicated in digit 8 of the second word; a 6 indicates a positive number, 7 a negative number. The following examples illustrate this conversion:

Before Conversion	*After Conversion*
+5212345678	@9592919293@9495969768
−4587654321	@9495989796@9594939271

3. *Automatic-Decimal, Double-Digit to Automatic-Decimal, Single-Digit.* Conversion is accompanied by sign sensing. The sign of TOFIELD will be set to plus if the next-to-last digit of FROMFIELD is different from 7, minus if it is 7. If the result is stored in part of a word, the sign of the entire word will be set to the sign of the result. The converse of the examples included under 1, above, illustrates this conversion.

4. *Floating-Decimal, Double-Digit to Floating-Decimal, Single-Digit.* FROMFIELD must occupy exactly two words; TOFIELD will occupy one word. Conversion is accompanied by sign sensing. The sign of TOFIELD will be set to plus if digit 8 of the second word of FROMFIELD is different from 7, minus if it is 7. The converse of the examples included under 2, above, illustrates this conversion.

5. *Automatic-Decimal, Old Format to Automatic-Decimal, New Format.* Four cases are distinguished:

SEQUENCE OF EDITING

FROMFIELD	TOFIELD	Automatic-decimal, single-digit to Automatic-decimal double-digit	Floating-decimal, single-digit to Floating-decimal, double-digit	Automatic-decimal, double-digit to Automatic-decimal, single-digit	Floating-decimal, double-digit to Floating-decimal, single-digit	Automatic-decimal, old format to Automatic-decimal, new format	Automatic-decimal to Floating-decimal	Floating-decimal to Automatic-decimal	Automatic-decimal double-digit to Print format	Floating-decimal, double-digit to Print format
		TYPE OF CONVERSION								
Automatic-decimal, single-digit	Automatic-decimal, single-digit					1				
	Automatic-decimal, double-digit	2				1				
	Floating-decimal, single-digit						1			
	Floating-decimal, double-digit		2				1			
	Automatic-decimal, print format	1							2	
	Floating-decimal, print format		2				1			3
Automatic-decimal, double-digit	Automatic-decimal, single-digit			1		2				
	Automatic-decimal, double-digit	3		1		2				
	Floating-decimal, single-digit			1			2			
	Floating-decimal, double-digit		3	1			2			
	Automatic-decimal, print format								1	
	Floating-decimal, print format		3	1			2			4
Floating-decimal, single-digit	Automatic-decimal, single-digit							1		
	Automatic-decimal, double-digit	2						1		
	Floating-decimal, double-digit		1							
	Automatic-decimal, print format	2						1	3	
	Floating-decimal, print format		1							2
Floating-decimal, double-digit	Automatic-decimal, single-digit				1			2		
	Automatic-decimal, double-digit	3			1			2		
	Floating-decimal, single-digit				1					
	Automatic-decimal, print format	3			1			2	4	
	Floating-decimal, print format									1

a. TOFIELD has more *decimal* places than FROMFIELD. These new decimal places will be filled with zeros. For example, a field whose automatic-decimal format is 3.2 is converted to a field whose format is 3.4. If the field contains 123.45 before conversion, it will contain 123.4500 after conversion.

b. TOFIELD has fewer *decimal* places than FROMFIELD. Extraneous decimals will be truncated after rounding. For example, a field whose automatic-demical format is 2.3 is converted to a field whose format is 2.2. If the field contains 55.467 before conversion, it will contain 55.47 after conversion; if it contains 55.464 before, it will contain 55.46 after.

c. TOFIELD has more *integer* places than FROMFIELD. These new integer places will be filled with high-order zeros. For example, a field whose automatic-decimal format is 2.3 is converted to a field whose format is 4.3. If the field contains 56.125 before conversion, it will contain 0056.125 after conversion.

d. TOFIELD has fewer *integer* places than FROMFIELD. A warning message will be issued during assembly that high-order digits may be lost. For example, a field whose automatic-decimal format is 4.1 is converted to a field whose format is 2.1. If the field contains 1545.7 before conversion, it will contain 45.7 after conversion.

Combinations of these conditions will cause all of the indicated actions to be taken.

6. *Automatic-Decimal to Floating-Decimal.* The first eight significant digits will be converted; others will be truncated without rounding. An automatic-decimal number is converted to a standard 7070 normalized, floating-decimal word. For example, −123.456789 is converted to −5312345678.

7. *Floating-Decimal to Automatic-Decimal.* Four cases are distinguished:

a. TOFIELD can accommodate the entire converted field. Any excess decimal places or integer places are filled with zeros. For example, if a field which contains +5287654321 is converted to a field whose automatic-decimal format is 4.7, the result will be +0087.6543210.

b. An integer of the converted number falls to the left of the high-order place of TOFIELD. This is an overflow condition, and the overflow latch on Accumulator 1 will be set ON. No warning message can be issued during assembly since this condition cannot be predicted on the basis of floating-decimal format alone. All digits that can be accommodated in their proper places will be stored. For example, if a field which contains −5412345678 is converted to a field whose automatic-decimal format is 3.5, the result will be −234.56780 and the overflow latch of Accumulator 1 will be turned ON.

c. The first digit of the converted number falls into one of the places of TOFIELD, but the decimals cannot be accommodated. Excess decimals will be truncated after rounding. For example, if a field which contains +4823456789 is converted to a field whose automatic-decimal format is 1.7, the result will be +0.0023457; if a field which contains −5398765432 is converted to a field whose format is 3.3, the result will be −987.654.

d. The first digit of the converted number falls to the right of the low-order place of the result field. Since the decimal value of the number is too small to register in the format of the result field, the field will be set equal to zero. For example, if a field which contains +3575757575 is converted to a field whose automatic-decimal format is 2.3, the result will be +00.000.

8. *Automatic-Decimal, Double-Digit to Print Format.* Editing is performed to fit data to a DLINE print image. (See page 65.) The necessary commas, decimal points, and other characters will be inserted.

9. *Floating-Decimal, Double-Digit to Print Format.* Editing is performed to convert a floating-decimal number to DLINE print format which is ± nn ±. xxxxxxxx, where ± nn is a two-digit exponent ±.xxxxxxxx is an eight-digit number. The value of the number is ±.xxxxxxxx multiplied by $10^{\pm nn}$. For example, @9591999897@9695949372 (Which is −5198765432 in single-digit form) will be printed as +01−.98765432, representing the number −.98765432 $\times 10^1$.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified.

BLANK PARAMETER *xx*

A parameter has been omitted. Its number will replace the *xx* of the message. A NOP will be generated.

HIGH-ORDER DIGITS LOST OF PARAMETER *xx*

The field in which the edited data is to be stored has a format that will cause loss of integer digits on the left. Such digits as can be accommodated in their proper places will be stored.

PRINT SUPPRESSED IF ALL ZERO

A print-line format has been written in such a way that all numerical positions are marked by Xs. A zero value for this field will result in no print at all.

TO-FIELD NOT ALPHA. PARAMETER *xx*

An attempt has been made to move an alphameric, mixed, or unspecified field to a field that is not alpha. The parameter number of the TOFIELD will replace the *xx* of the message. The field will be moved but not edited.

UNACCEPTABLE PARAMETER *xx*

The *xx* will be replaced by the number of a parameter that is not one of the types listed as acceptable under "Source Program Format." A NOP will be generated.

Examples

The following are examples of acceptable coding for the EDMOV macro-instruction. For each, the associated source-program entries are given, followed by the EDMOV statement coding generated in-line and coding generated out-of-line.

```
PAGE AA    PROGRAM                                        7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION     REF

01  701    *                 EDMOV EXAMPLE 1
02  702    SOMELABEL  DLINE                                                 00001            +0003250328
03  703    FIELDB            10X,XXX.ZZ)                                          89  0326                  0326
04  704               DA     1                                                               +0003290329
05  705    FIELDA            00,05A4.2                                            05  0329                  0329
06  706    *
07  707    ANYLABEL   EDMOV  FIELDA TO FIELDB
08        X ANYLABEL  ZA2    FIELDA(0,5)                                    00002     0330 +2300050329
09        X           ST2    COMAREA.A+2                                              0331 +2200090352
10        X           XZA    MACREG.01,COMAREA.A+2                                    0332 +4600010352
11        X           ENB    MACREG.01,EDMOV02.A                                      0333 +5700010349
12        X           ZA3    COMAREA.A(8,9)                                           0334 +3300890350
13        X           ST3    FIELDB(0,1)                                    00003     0335 +3200890326
14        X           BZ3    *+2                                                      0336 +3000090338
15        X           ZA3    ','                                                      0337 +3300230353
16        X           ST3    FIELDB(2,3)                                              0338 +3200010327
17        X           ZA3    COMAREA.A(10,15)                                         0339 +3300050351
18        X           ST3    FIELDB(4,9)                                    00004     0340 +3200270327
19        X           BZ3    *+2                                                      0341 +3000090343
20        X           ZA3    '.'                                                      0342 +3300010353
21        X           ST3    FIELDB(10,11)                                            0343 +3200890327
22        X           ZA3    COMAREA.A(16,19)                                         0344 +3300690351
23        X           ST3    FIELDB(12,15)                                  00005     0345 +3200030328
24        X           ZA3    '9'                                                      0346 +3300450353
25        X           ST3    FIELDB(12,12)                                            0347 +3200000328
26        X           ST3    FIELDB(14,14)                                            0348 +3200220328
27  708    *
28  709    *        THE FOLLOWING IS GENERATED OUT OF LINE
29  710    *
30        X EDMOV02.A  DRDW  -COMAREA.A,COMAREA.A+1                                   0349 -0003500351
31        X COMAREA.A  DA                                                                  +0003500352
                      LITERALS
32        X                  '.'                                            00006 01  0353 '15              0353
33        X                  ','                                                  23  0353 '  35            0353
34        X                  '9'                                                  45  0353 '     99         0353
```

EDMOV Example 1

The automatic-decimal field FIELDA is edited to the print format specified in the DLINE entry.

```
PAGE AA     PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN  CDREF  LABEL        OP     OPERAND                                          CDNO  FD   LOC  INSTRUCTION     REF

01   714   *                   EDMOV EXAMPLE 2
02   715                DA     1                                                                +0003250325
03   716   AUTODECNO           00,09A8.2                                              09   0325                  0325
04   717                DA     1                                                                +0003260326
05   718   FLTPTNO             00,09F                                                 09   0326                  0326
06   719   *
07   720   ANYLABEL     EDMOV  AUTODECNO TO FLTPTNO
08        X ANYLABEL    ZA1    +0                                               00001      0327 +1300000341
09        X             ZA2    AUTODECNO(0,9)                                              0328 +2300090325
10        X             ZA3    +0000000068                                                 0329 +3300090340
11        X             BLX    94,FLOT2.A                                                  0330 +0200940334
12        X             ZST1   FLTPTNO                                                     0331 -1100090326
13   721   *
14   722   *         THE FOLLOWING IS GENERATED OUT OF LINE
15   723   *
16        X FLOT1.A     SLC1   MACREG.1                                         00002      0332 +5000011300
17        X             B      *+2                                                         0333 +0100090335
18        X FLOT2.A     SLC    MACREG.1                                                    0334 -5000010300
19        X             BZ1    0+X94                                                       0335 +1094090000
20        X             S3     MACREG.1(4,5)                                               0336 -3400450001
21        X             SR1    2                                                00003      0337 +5000001002
22        X             STD3   9991(0,1)                                                   0338 -3200019991
23        X FLOT3.A     B      0+X94                                                       0339 +0194090000
                        LITERALS
24        X                    +0000000068                                        09   0340 +0000000068      0340
25        X                    +0                                                 00   0341 +0               0341
```

EDMOV Example 2

The automatic-decimal number is edited to the floating-decimal format.

```
PAGE AA     PROGRAM                                  7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION       REF

01  727   *                  EDMOV EXAMPLE 3.
02  728   SOMELABEL   DLINE                                                 00001              +0003250328
03  729   AMT                10$X,XXX.ZZ)DR,CR                                    89  0326                  0326
04  730               DA     2,,0+INDEXWORD                                                    +0003290330
05  731   AMTFIELD           3,9A4.3                                              39  0329                  0000
06  732   *
07  733   ANYLABEL    EDMOV  AMTFIELD TO AMT
08          X ANYLABEL ZA2   AMTFIELD(0,6)+INDEXWORD                        00002     0331 +2301390000
09          X         SRR2   1                                                        0332 +5000002101
10          X         BM2    M.6                                                      0333 -2000090362
11          X         ZA3    '    DR'                                                 0334 +3300090377
12          X M.3     ST3    AMT(18,21)                                               0335 +3200690328
13          X         ST2    COMAREA.A+2                                    00003     0336 +2200090374
14          X         XZA    MACREG.01,COMAREA.A+2                                    0337 +4600020374
15          X         ENA    MACREG.01,EDMOV02.A                                      0338 +5600020364
16          X         ZA3    '     '                                                  0339 +3300090375
17          X         ST3    AMT(0,1)                                                 0340 +3200890326
18          X         ST3    AMT(2,9)                                       00004     0341 +3200070327
19          X         SLC2   MACREG.02                                                0342 +5000032300
20          X         ZA3    '$'                                                      0343 +3300230378
21          X         B      M.4-4+MACREG.02                                          0344 +0103090361
22          X M.5     ST3    AMT(0,1)                                                 0345 +3200890326
23          X         ZA3    COMAREA.A(8,9)                                 00005     0346 +3300890372
24          X         ST3    AMT(2,3)                                                 0347 +3200010327
25          X         ZA3    ','                                                      0348 +3300450378
26          X         ST3    AMT(4,5)                                                 0349 +3200230327
27          X         ZA3    COMAREA.A(10,11)                                         0350 +3300010373
28          X         ST3    AMT(6,7)                                       00006     0351 +3200450327
29          X         ZA3    COMAREA.A(12,13)                                         0352 +3300230373
30          X         ST3    AMT(8,9)                                                 0353 +3200670327
31          X         ZA3    COMAREA.A(14,15)                                         0354 +3300450373
32          X         ST3    AMT(10,11)                                               0355 +3200890327
33          X         ZA3    '.'                                            00007     0356 +3300010378
34          X         ST3    AMT(12,13)                                               0357 +3200010328
35          X         ZA3    COMAREA.A(16,17)                                         0358 +3300670373
36          X         ST3    AMT(14,15)                                               0359 +3200230328
37          X         ZA3    COMAREA.A(18,19)                                         0360 +3300890373
38          X         ST3    AMT(16,17)                                     00008     0361 +3200450328
39  734   *
40  735   *        THE FOLLOWING IS GENERATED OUT OF LINE
41  736   *
42          X M.6     ZA3    '    CR'                                                 0362 +3300090376
43          X         B      M.3                                                      0363 +0100090335
44          X EDMOV02.A DRDW -COMAREA.A,COMAREA.A+1                                   0364 -0003720373
45          X M.4     B      M.5                                                      0365 +0100090345
46          X         B      M.5+4                                          00009     0366 +0100090349
47          X         B      M.5+6                                                    0367 +0100090351
48          X         B      M.5+8                                                    0368 +0100090353

PAGE AB     PROGRAM                                                                                       PAGE AB

LN CDREF  LABEL       OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION       REF

01          X         B      M.5+10                                                   0369 +0100090355
02          X         B      M.5+10                                                   0370 +0100090355
03          X         B      M.5+10                                         00010     0371 +0100090355
04          X COMAREA.A DA                                                                 +0003720374
                      LITERALS
05          X                '     '                                        00011 09  0375 '0000000000    0375
06          X                '    CR'                                             09  0376 '0000006379    0376
07          X                '    DR'                                             09  0377 '0000006479    0377
08          X                '.'                                                  01  0378 '15            0378
09          X                '$'                                                  23  0378 '  25          0378
10          X                ','                                                  45  0378 '    35        0378
```

EDMOV Example 3

An automatic-decimal number is edited to the print format with a floating dollar sign and debit and credit indication.

MOVE — Move Data

MOVE generates instructions that will transmit data from one specified field or area in storage to another.

Source Program Format

The basic formats for the MOVE statement in the source program are as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–) Basic Autocoder
01	ANYLABEL	MOVE	THISFIELD TO THATFIELD
02	ANYLABEL	MOVE	FIELDA,FIELDB,etc. TO THATFIELD
03	ANYLABEL	MOVE	THISFIELD TO FIELDD,FIELDE,etc.
04	ANYLABEL	MOVE	S TO T AND U,V,etc. TO W AND X TO Y,Z,etc.
05			

In these examples, ANYLABEL is any symbolic label; it may be omitted. The entry MOVE must appear exactly as written. THISFIELD, FIELDA, FIELDB, S, U, V, and X are either the symbolic names of the fields or areas to be moved or alphameric or numerical literals. Numerical literals must be signed. TO and AND are operators that must be written exactly as shown, preceded and followed by a blank. THATFIELD, FIELDD, FIELDE, T, W, Y, and Z are storage locations to which the data is to be moved; the addresses must be symbolic names of fields or areas.

If there are several "from" fields (as in the second format) or several "to" fields (as in the third format), they must be separated by commas. It is not possible to move multiple "from" fields to multiple "to" fields; an attempt to do so will result in an error condition.

Data may bridge words and start at any position in a word, both in the "from" fields and in the "to" fields. Symbolically referenced fields may be any length; literals are restricted as indicated under "Limitations on Length." Data characteristics do not affect the transmission. Data will always be left-justified in the field(s) to which they are moved. The sign of the last item stored in any location determines the sign of the entire word.

In the first format, if THISFIELD is larger than THATFIELD, movement of data will be terminated when THATFIELD is filled. If THISFIELD is smaller than THATFIELD, the data from THISFIELD will be left-justified in THATFIELD and the remaining portion of THATFIELD filled with zeros. The sign of these zero words will be the same as that of the last word moved.

In the second format, the data in FIELDA will be moved to THATFIELD and left-justified. Data from FIELDB will be entered beginning with the digit position following the one in which data from FIELDA terminated. The movement of data continues in the same fashion until the contents of all the specified "from" fields have been moved or until THATFIELD is filled. If all the "from" fields have been moved before THATFIELD is filled, the remainder will be filled with zeros. The sign of the zero words will be the same as that of the last word moved.

In the third format, the data in THISFIELD will be moved to FIELDD and to each subsequent field until all such fields are filled or THISFIELD has been completely

transferred. If the data in THISFIELD is accommodated in the "to" fields without filling them, the remainder of the field(s) will be zeroed out; the sign of the zero words will be the same as that of the last word moved.

In the fourth format, several MOVE operations are performed. The MOVE operations are linked by the operator AND, as indicated. Any of the above three formats for the MOVE statement may be used.

Processing Techniques

Limitations on Length

The operand portion of the MOVE macro-instruction may contain 75 parameters. The operators TO and AND are counted as parameters. No limitation is placed on the size of the field if it is referenced symbolically. Literals are restricted as follows: alphameric literals, 120 characters; automatic-decimal literals, 20 digits.

Address Modification

All symbolic addresses may be modified by indexing and address adjustment.

The Effect of MOVE

The MOVE macro-instruction is non-destructive in that it does not alter the contents of the "from" field(s). A possible exception might be the case in which the "to" field(s) begin to overlay the "from" field(s), in which case a strict left-to-right procedure of data movement would be maintained.

The following examples illustrate the effect of MOVE on various fields.

No.	*"From" field(s)*	*"To" field(s) before MOVE*	*"To" field(s) after MOVE*
1	+1234567890+12345	−6857463590−7948375403	+1234567890+1234500000
2	+1111111111+22−3333	−9999999999+9999999999	+1111111111−2233330000
3	+1111111111	−99999−9999−9999	+11111+1111+1000
4	−9876543210	+55555	−98765
5	+8888888888+8888	+0000000000−00	+8888888888+88
6	@6162636465	+123456789	@616263646
7	@717273	+1234567	@7172730
8	@818283	+123+123	@818@283

Examples 1, 4, 6, and 7 illustrate a single "from" field and a single "to" field. Examples 2 and 5 illustrate multiple "from" fields. Examples 3 and 8 illustrate multiple "to" fields. In examples 6, 7, and 8, machine difficulty may arise when an attempt is made to print out the "to" fields since invalid double-digit combinations were created at the end of each field by the MOVE.

Although the MOVE macro-instruction may refer to the label(s) of *any* field or area, it is most frequently used in conjunction with input/output macro-instructions. For example, records (other than Form 3 or Form 4 records as described in the bulletin "IBM 7070 Input/Output Control System") that have undergone preliminary processing in the input area may be moved to a work area by means of a MOVE macro-instruction that references the labels of the DA header lines of the input area and the work area. As another example, a print line might be included in a tape output file by following a PUT macro-instruction by a MOVE macro-instruction that references the label of the DLINE header line and the label of the DA header line of the output area.

When MOVE references the label of a declarative statement other than DA or DRDW, coding will be generated to cause, as a maximum, the entire area or constant defined to be moved or filled.

When MOVE references the label of a DA header line which does not specify a relative address and implicit indexing, coding will be generated to cause, as a maximum, the *first* record area defined to be moved or filled.

When MOVE references the label of a DA header line which specifies a relative address and implicit indexing, coding will be generated to cause, as a maximum, the *current* record area (as determined by the contents of the implicit index word) to be moved or filled.

If MOVE references the label of a DRDW, coding will be generated to move the RDW only, *not* the area it defines.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

MULTIPLE FROM- AND TO-FIELDS

An attempt has been made to move data from several fields to several others, which is a format violation. A NOP will be generated.

NO FIELD SIZE. PARAMETER *xx*

The *xx* of the message is replaced by the number of a parameter for which the record does not indicate its size. Thus no coding can be generated to move its contents or to store data in it. A NOP is generated.

NO FROM-FIELD IN MOVE MACRO LINE

The first parameter in the operand portion is the operator TO. A NOP is generated.

NO TO-FIELD IN MOVE MACRO LINE

No parameter follows the operator TO. A NOP is generated.

TO-FIELD(S) SMALLER THAN FROM-FIELD(S)

The field(s) to which data is to be moved cannot accommodate all of the data of the "from" field(s). Transmission terminates when the "to" area is filled.

UNACCEPTABLE PARAMETER *xx*

The *xx* of the message will be replaced by the number of a parameter that is not of the types listed under "Source Program Format" as valid. A NOP is generated.

Examples

The following are examples of acceptable coding for the MOVE macro-instruction. For each, the associated source-program entries are given, followed by the MOVE statement, coding generated in-line, and (where applicable) coding generated out-of-line.

```
PAGE AA     PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA
LN CDREF  LABEL        OP     OPERAND                                  CDNO FD  LOC  INSTRUCTION     REF
01  740   *                   MOVE EXAMPLE 1
02  741               DA      1                                                     +0003250325
03  742   HERE                00,09A                                         09  0325                0325
04  743               DA      1                                                     +0003260326
05  744   THERE               00,09A                                         09  0326                0326
06  745   *
07  746   ANYLABEL    MOVE    HERE TO THERE
08      X ANYLABEL    ZA2     HERE(0,9)                                00001     0327 +2300090325
09      X             ST2     THERE(0,9)                                         0328 +2200090326
10  747   *
```

MOVE Example 1

A ten-digit, automatic-decimal number is moved from one location to another location of the same size.

```
PAGE AA     PROGRAM                                    7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA
LN CDREF  LABEL        OP     OPERAND                                  CDNO FD  LOC  INSTRUCTION     REF
01  751   *                   MOVE EXAMPLE 2
02  752               DA      1                                                     +0003250339
03  753   HERE                00,149'                                        09  0325                0325
04  754               DA      1                                                     +0003400366
05  755   AA                  00,49'                                         09  0340                0340
06  756   BB                  80,139'                                        09  0348                0348
07  757   CC                  230,269'                                       09  0363                0363
08  758   *
09  759   ANYLABEL    MOVE    HERE TO AA,BB,CC
10      X ANYLABEL    XZA     MACREG.1,HERE                            00001     0367 +4600010325
11      X             RS      MACREG.1,M.1                                       0368 +6500010369
12  760   *
13  761   *        THE FOLLOWING IS GENERATED OUT OF LINE
14  762   *
15      X M.1         DRDW    +AA,AA+4                                           0369 +0003400344
16      X M.3         DRDW    +BB,BB+5                                           0370 +0003480353
17      X M.5         DRDW    -CC,CC+3                                           0371 -0003630366
```

MOVE Example 2

A 75-character (150-digit) alphameric field is moved to three fields, defined under a single DA, whose total storage area is also equal to 150 digits.

```
PAGE AA     PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA
LN CDREF   LABEL      OP    OPERAND                                                   CDNO FD   LOC  INSTRUCTION       REF
01  766    *                MOVE EXAMPLE 3
02  767    FIELDS     DA    1                                                                        +0003250326
03  768    EE               00,07A                                                         07   0325                   0325
04  769    FF               08,15A                                                         89   0325                   0325
05  770    *
06  771    ANYLABEL   MOVE  FF TO EE
07       X ANYLABEL   ZA2   FF(0,1)                                                  00001      0327 +2300890325
08       X            SL    6                                                                   0328 -5000000206
09       X            A2    FF(2,7)                                                             0329 +2400050326
10       X            ST2   EE(0,7)                                                             0330 +2200070325
11  772    *
```

MOVE Example 3

An eight-digit field that bridges two locations is moved to a field that does not bridge locations.

```
PAGE AA     PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA
LN CDREF   LABEL      OP    OPERAND                                                   CDNO FD   LOC  INSTRUCTION       REF
01  776    *                MOVE EXAMPLE 4
02  777    FIELDS     DA    1                                                                        +0003250347
03  778    AA               00,19A                                                         09   0325                   0325
04  779    BB               20,99A                                                         09   0327                   0327
05  780    CC               100,229A                                                       09   0335                   0335
06  781               DA    1                                                                        +0003480371
07  782    THERE            00,239A                                                        09   0348                   0348
08  783    *
09  784    ANYLABEL   MOVE  AA,BB,CC TO THERE
10       X ANYLABEL   XZA   MACREG.1,THERE                                           00001      0372 +4600010348
11       X            RG    MACREG.1,M.1                                                        0373 -6500010377
12       X            ZA2   THERE(229,229)                                                      0374 +2300990370
13       X            SL    20                                                                  0375 -5000000220
14       X            ST2   THERE(230,239)                                                      0376 +2200090371
15  785    *
16  786    *         THE FOLLOWING IS GENERATED OUT OF LINE
17  787    *
18       X M.1        DRDW  +AA,AA+1                                                 00002      0377 +0003250326
19       X M.3        DRDW  +BB,BB+7                                                            0378 +0003270334
20       X M.5        DRDW  -CC,CC+12                                                           0379 -0003350347
```

MOVE Example 4

Three automatic-decimal fields are moved to a single, larger field. The unoccupied portion of the "to" field is filled with zeros.

```
PAGE AA    PROGRAM                                7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA
LN CDREF  LABEL      OP    OPERAND                                     CDNO  FD  LOC   INSTRUCTION   REF

01  801   *                MOVE EXAMPLE 5
02  802   HERE       DA    1                                                          +0003250339
03  803                    00,149'                                           09  0325                0325
04  804              DA    1                                                          +0003400344
05  805   THEREA           00,49'                                            09  0340                0340
06  806              DA    1                                                          +0003450352
07  807   THEREB           00,75'                                            09  0345                0345
08  808              DA    1                                                          +0003530355
09  809   THEREC           00,23'                                            09  0353                0353
10  810   *
11  811   ANYLABEL   MOVE  HERE TO THEREA,THEREB,THEREC
12      X ANYLABEL   XZA   MACREG.1,HERE                               00001     0356 +4600010325
13      X            RS    MACREG.1,M.1                                          0357 +6500010368
14      X            ZA2   HERE(120,125)                                         0358 +2300050337
15      X            ST2   THEREB(70,75)                                         0359 +2200050352
16      X            ZA1   HERE(126,129)                                         0360 +1300690337
17      X            ZA2   HERE(130,139)                               00002     0361 +2300090338
18      X            SL    6                                                     0362 -5000000206
19      X            A2    HERE(140,145)                                         0363 +2400050339
20      X            ST2   THEREC(10,19)                                         0364 +2200090354
21      X            ST1   THEREC(0,9)                                           0365 +1200090353
22      X            ZA2   HERE(146,149)                               00003     0366 +2300690339
23      X            ST2   THEREC(20,23)                                         0367 +2200030355
24  812   *
25  813   *        THE FOLLOWING IS GENERATED OUT OF LINE
26  814   *
27      X M.1        DRDW  +THEREA,THEREA+4                                      0368 +0003400344
28      X M.3        DRDW  -THEREB,THEREB+6                                      0369 -0003450351
```

MOVE Example 5

A 75-character (150-digit) alphameric field is moved into three fields. The fields are each defined by subsequent entries of separate DAs and their total storage area is equal to 150 digits. However, the RDWs generated reserve an area of 160 digits. The possibility of invalid alpha combinations exists in digit positions 76, 79 of THEREB and in digit positions 24, 29 of THEREC since these segments are not affected by MOVE.

```
PAGE AA     PROGRAM                                      7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01  818   *                 MOVE EXAMPLE 6
02  819              DA     1                                                              +0003250326
03  820   HEREA             00,19A                                                09  0325                  0325
04  821              DA     1                                                              +0003270329
05  822   HEREB             00,29A                                                09  0327                  0327
06  823              DA     1                                                              +0003300332
07  824   HEREC             00,24A                                                09  0330                  0330
08  825   THERE      DA     1                                                              +0003330340
09  826                     74                                                    44  0340                  0340
10  827   *
11  828   ANYLABEL   MOVE   HEREA,HEREB,HEREC TO THERE
12      X ANYLABEL   XZA    MACREG.1,THERE                                   00001      0341 +4600010333
13      X            RG     MACREG.1,M.1                                                0342 -6500010345
14      X            ZA2    HEREC(20,24)                                                0343 +2300040332
15      X            ZST2   THERE(70,74)                                                0344 -2100040340
16  829   *
17  830   *       THE FOLLOWING IS GENERATED OUT OF LINE
18  831   *
19      X M.1        DRDW   +HEREA,HEREA+1                                              0345 +0003250326
20      X M.3        DRDW   +HEREB,HEREB+2                                   00002      0346 +0003270329
21      X M.5        DRDW   -HEREC,HEREC+1                                              0347 -0003300331
```

MOVE Example 6

Three automatic-decimal fields, each defined by a separate DA and with a total storage area of 75 digits, are moved to an area of equal size but with unspecified characteristics.

```
PAGE AA     PROGRAM                                      7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01  835   *                         MOVE EXAMPLE 7
02  836              DA     5,,0+INDEXWORD                                                 +0003250329
03  837   HERE              00,09A                                                09  0325                  0000
04  838              DA     1                                                              +0003300330
05  839   THERE             00,09A                                                09  0330                  0330
06  840   *
07  841   ANYLABEL   MOVE   HERE TO THERE
08      X ANYLABEL   ZA2    HERE(0,9)+INDEXWORD                              00001      0331 +2301090000
09      X            ST2    THERE(0,9)                                                  0332 +2200090330
10  842   *
```

MOVE Example 7

The MOVE statement references the label of a subsequent entry under a DA that contains a relative address and implicit indexing. The contents of the HERE field of the current record (as determined by the contents of the implicit index word) will be moved to the THERE field.

```
PAGE AA    PROGRAM                                                  7070 COMPILER SYSTEM VERSION MASTR, CHANGE LEVEL 00284.

LN   LABEL       OP     OPERAND                                     LOC  INSTRUCTION REF.ADR PGLIN CD NO

01   *                MOVE EXAMPLE 8                                                          846
02               DA     2,,0+INDEXWORD                                   +0003250326          847
03   HERE               00,09A                                      0325            09 0000   848
04               DA     2,,0+X15                                         +0003270330          849
05   THERE              00,12A                                      0327            09 0000   850
06   *                                                                                        851
07   ANYLABEL    MOVE   HERE TO THERE                                                         852
08   ANYLABEL    ZA2    HERE(0,9)+(INDEXWORD                        0331 +2301090000          GENRD 00001
09               ST2    THERE(0,9)+X15                              0332 +2215090000          GENRD
10               SL     20                                          0333 -5000000220          GENRD
11               ST2    THERE(10,12)+X15                            0334 +2215020001          GENRD
```

MOVE Example 8

The ten-digit HERE field of the current record is moved to the larger THERE field of the current record. The unoccupied portion of the THERE field will be filled with zeros and the sign will be made the same as the sign of the last word moved.

```
PAGE AA    PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL       OP     OPERAND                                        CDNO FD  LOC  INSTRUCTION      REF

01 8501   *                MOVE EXAMPLE 9
02 8502   HERE        DA     1                                                             +0003250328
03 8503   AA                 00,15A                                              09  0325                  0325
04 8504   BB                 20,25'                                              05  0327                  0327
05 8505   CC                 26,32                                               69  0327                  0327
06 8506   THERE       DA     1                                                             +0003290333
07 8507                      42                                                  22  0333                  0333
08 8508   *
09 8509   ANYLABEL    MOVE   HERE TO THERE
10      X ANYLABEL    XZA    MACREG.1,HERE                                  00001    0334 +4600010325
11      X             RS     MACREG.1,M.1                                            0335 +6500010339
12      X             ZA2    THERE(39,39)                                            0336 +2300990332
13      X             SL     20                                                      0337 -5000000220
14      X             ST2    THERE(40,49)                                            0338 +2200090333
15 8510   *
16 8511   *        THE FOLLOWING IS GENERATED OUT OF LINE
17 8512   *
18      X M.1         DRDW   -THERE,THERE+3                                 00002    0339 -0003290332
```

MOVE Example 9

The entire HERE field through digit position 39 is moved to the larger THERE field. The unoccupied portion of the THERE field (40, 49) will be filled with zeros and the sign will be made the same as the sign of the last word moved. In this example, the possibility of creating invalid alpha combinations does not exist since the sign of the word to which field BB and part of field CC are to be moved is determined by the sign of CC. Since the format of field CC is unspecified, the processor assumes the field contains numerical data to be treated as a signed integer.

SHIFT—Shift and Store

SHIFT generates instructions to place the contents of a field into one or more accumulators, to shift the data in a specified way, and to store the result.

Source Program Format

The basic format for the SHIFT statement in the source program is as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45 50 Basic Au
01	ANYLABEL	SHIFT	OPTION, START, COUNT, FIELDA, FIELDB
02			

ANYLABEL is any symbolic label; it may be omitted. The entry SHIFT must be written exactly as shown. OPTION is one of the following one- or two-letter codes, specifying the type of shift to be executed:

OPTION	*Type of Shift*
L	Left
R	Right
LC	Left and Count
RR	Right and Round
LS	Left Split
RS	Right Split

START is an integer indicating where to begin shifting in case of a split-shift option; it is determined by counting from the left-most digit of the field, beginning with 1. (With other options, START will be blank, but the separating commas must be entered.) With one exception, COUNT is the actual number of positions to be shifted, restricted only by the size of the field. In the case of a shift-left-and-count option, however, COUNT must be an index word, referenced either by its actual, two-digit number, without a preceding X, or by its symbolic name, which will contain in digit positions 4-5 the number of high-order zeros found in the shifted field. FIELDA is the symbolic address of the field to be shifted; it must be less than twenty-one digits in length. FIELDB is the field in which the result is to be stored; it may also be referenced by its symbolic address. The sign of the field to be shifted is transmitted with the field and stored with the result in FIELDB. The basic format may be modified by the omission of FIELDB. In that case, the data will be restored in its original field after shifting.

No warning is issued either in case of possible digit loss when the result of a shift is larger than FIELDB or when invalid alpha combinations have been created.

Processing Techniques

Limitations on Length

The number of parameters is fixed by the format. The field to be shifted is limited to twenty-digit length.

Address Modification

All symbolic addresses may be modified by indexing and address adjustment.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

COUNT GREATER THAN FIELD-SIZE

The COUNT field contains an integer larger than the number of digits in the field to be shifted.

COUNT IS ZERO

No shift will be carried out. The field will be stored without shifting.

COUNT NOT AN INDEXWORD

In a shift-left-and-count option, a label has been entered for COUNT that is defined elsewhere as other than an index word. A NOP will be generated.

FIELD GREATER THAN 20 DIGITS

An error condition, since fields to be shifted may not exceed this size. A NOP will be generated.

FIELD UNACCEPTABLE

An attempt has been made to shift an alteration switch or some other entity that is not a part of storage. A NOP will be generated.

INCORRECT NUMBER OF PARAMETERS

An attempt has been made to write more than five parameters into the operand portion of a SHIFT statement. Parameters in excess of the first five will be ignored.

INCORRECT OPTION

The option code is blank or is not one of those listed under "Source Program Format."

SHIFTING INSTRUCTION

The field to be shifted is an imperative instruction. Coding to accomplish this will be generated nevertheless.

START GREATER THAN FIELD-SIZE

The START counter in a split-shift option contains an integer larger than the number of digits in the field to be shifted.

Examples

The following are examples of acceptable coding for the SHIFT macro-instruction. For each, the associated source-program entries are given, followed by the SHIFT statement, coding generated in-line, and (where applicable) coding generated out-of-line.

```
PAGE AA     PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                                   CDNO FD  LOC  INSTRUCTION      REF

01   856  *                 SHIFT EXAMPLE 1
02   857             DA     1                                                                        +0003250325
03   858  FIELDA            00,09                                                          09  0325                  0325
04   859  *
05   860  ANYLABEL   SHIFT  L,,3,FIELDA
06      X ANYLABEL   ZA2    FIELDA(0,9)                                               00001     0326 +2300090325
07      X            SL2    3                                                                   0327 +5000002203
08      X            ST2    FIELDA(0,9)                                                         0328 +2200090325
09   861  *
```

SHIFT Example 1

The contents of FIELDA are shifted left three positions and the result is stored again in FIELDA.

```
PAGE AA     PROGRAM                                          7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.   PAGE AA

LN CDREF  LABEL      OP     OPERAND                                                   CDNO FD  LOC  INSTRUCTION      REF

01   865  *                 SHIFT EXAMPLE 2
02   866  COUNT      EQU    90,X
03   867             DA     1                                                                        +0003250325
04   868  FIELDA            00,09                                                          09  0325                  0325
05   869             DA     1                                                                        +0003260326
06   870  FIELDB            00,09                                                          09  0326                  0326
07   871  *
08   872  ANYLABEL   SHIFT  LC,,COUNT,FIELDA,FIELDB
09      X ANYLABEL   ZA2    FIELDA(0,9)                                               00001     0327 +2300090325
10      X            SLC2   COUNT                                                               0328 +5000902300
11      X            ST2    FIELDB(0,9)                                                         0329 +2200090326
12   873  *
```

SHIFT Example 2

The digits in FIELDA are shifted to the left until a digit other than a zero is in the high-order position. If the high-order digit is non-zero to start with, no shift takes place. The number of positions shifted is recorded in the index word labeled COUNT.

```
PAGE AA     PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF   LABEL       OP     OPERAND                                               CDNO FD  LOC  INSTRUCTION      REF

01  874    *                  SHIFT EXAMPLE 3
02  875               DA      1                                                                   +0003250328
03  876    FIELDA             00,14                                                      09  0325                  0325
04  877    FIELDB             20,34                                                      09  0327                  0327
05  878    *
06  879    ANYLABEL   SHIFT   RR,,5,FIELDA,FIELDB
07       X ANYLABEL   ZA1     +0                                                  00001      0329 +1300000337
08       X            ZA2     FIELDA(0,9)                                                    0330 +2300090325
09       X            SL      5                                                              0331 -5000000205
10       X            A2      FIELDA(10,14)                                                  0332 +2400040326
11       X            SRR     5                                                              0333 -5000000105
12       X            ST2     FIELDB(10,14)                                       00002      0334 +2200040328
13       X            SR      5                                                              0335 -5000000005
14       X            ST2     FIELDB(0,9)                                                    0336 +2200090327
15  880    *
16  881    *         THE FOLLOWING IS GENERATED OUT OF LINE
17  882    *
                      LITERALS
18       X                    +0                                                         00  0337 +0               0337
```

SHIFT Example 3

The contents of FIELDA are shifted right 5 positions and the amount shifted is rounded off. The result is stored in FIELDB.

```
PAGE AA     PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF   LABEL       OP     OPERAND                                               CDNO FD  LOC  INSTRUCTION      REF

01  885    *                  SHIFT EXAMPLE 4
02  886               DA      1                                                                   +0003250328
03  887    FIELDA             00,19                                                      09  0325                  0325
04  888    FIELDB             20,39                                                      09  0327                  0327
05  889    *
06  890    ANYLABEL   SHIFT   RS,12,8,FIELDA,FIELDB
07       X ANYLABEL   ZA1     FIELDA(0,9)                                         00001      0329 +1300090325
08       X            ZA2     FIELDA(10,19)                                                  0330 +2300090326
09       X            SRS     8(11)                                                          0331 -5000001608
10       X            ST2     FIELDB(10,19)                                                  0332 +2200090328
11       X            ST1     FIELDB(0,9)                                                    0333 +1200090327
12  891    *
```

SHIFT Example 4

Starting with the 12th digit, the contents of FIELDA are shifted 8 positions to the right. The result is stored in FIELDB.

SNAP—Memory Print-Out

SNAP generates instructions to provide a listing of a specified portion of storage.

Source Program Format

The basic format for the SNAP statement in the source program is:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–60) Basic Autocoder →	Autocoder → (61–75)
01	ANYLABEL	SNAP	FROMADDR,TOADDR,OUTPUTUNIT,SKIPSPRIOR,SN	APS,SKIPSAFTER
02				

ANYLABEL is any symbolic label; it may be omitted. The entry SNAP must be written exactly as shown. FROMADDR and TOADDR represent the limits of storage to be listed and may be symbolic or actual addresses. OUTPUTUNIT is the symbolic address of any tape channel and unit, punch, printer, or typewriter that has been defined by an EQU statement elsewhere in the program. SKIPSPRIOR, SNAPS, and SKIPSAFTER are unsigned, one- or two-digit integer counters. SKIPSPRIOR controls the number of times the program will pass through the location of the SNAP statement without taking a print-out. SNAPS controls the number of times following this that print-outs will be taken. SKIPSAFTER controls the number of times that print-outs will again be omitted. Thereafter, control alternates between SNAPS and SKIPSAFTER for the duration of the program.

The basic format may be modified by the omission of one or more of the three counters. If omitted, SKIPSPRIOR will automatically be set equal to 0, SNAPS to 1, and SKIPSAFTER will cause permanent skipping once the specified number of SNAPS has been taken. If either SKIPSPRIOR or SNAPS is omitted, or both, separating commas must be punched.

Processing Techniques

Limitations on Length

The number of parameters is fixed by the basic format, subject only to the omission of one or more counters. There is no limit to the amount of storage to be printed out, although approximately 150 locations of storage will be required by the calling sequence and the subroutine that constitute SNAP. Conceivably these may be positioned so as to fall between FROMADDR and TOADDR, in which case this area of storage will be shown as required by the SNAP instruction. The programmer has control over the placement of the subroutine, however, by means of the LITORIGIN statement.

Address Modification

All addresses may be modified by indexing; symbolic addresses may be modified by address adjustment.

The Effect of SNAP

The print-out provided by SNAP is non-destructive and will consist of the electronic switches, the accumulators, index words 10 and 11, and storage between the limits specified. (In the case of a type-out, only storage will be given.) The stor-

age area specified must be contiguous. Index words 10 and 11 are used by SNAP; if they fall between FROMADDR and TOADDR, their representation in that section of the print-out will be that required by SNAP, not that given by the source program. For this reason, the source-program contents of these words are printed out ahead of the storage area. Before the exit from the SNAP subroutine, both index words are restored to their original contents.

The output will be in the following format:

First line:	SNAP Electronic Switches, Instruction Counter at time of SNAP branch.
Second line:	Accumulators, Index Words 10 and 11.
Each succeeding line:	Five words of storage each, with the location of the first word printed on each line shown for identification purposes. Numerical words carry the proper sign. Alpha words are printed with an A in the sign position and contents in double-digit representation; this is done to avoid printer difficulty which might arise if an alpha location contained a double-digit combination that was not the code for any character.

When tape output is specified, tape density must have been set previously by the program or the operator. No provision for tape labels, end-of-file conditions, or tape mark and rewind routines has been made in order to conserve space, and because it will often be desirable to use the same output tape for SNAP print-outs as that used for utility program output.

When the SNAP routine is entered, the priority mask is set to "prohibit." On leaving the subroutine, the mask is set to "allow," regardless of its prior condition. If the programmer does not want the mask at "allow," he must restore it after each use of SNAP.

Error and Warning Messages

The following error and warning messages will be issued during assembly under the conditions specified:

BLANK PARAMETER *xx*

The *xx* of the message will be replaced by 01, 02, or 03, depending upon whether FROMADDR, TOADDR, or OUTPUTUNIT has been left blank. A NOP will be generated.

INVALID PARAMETER *xx*

The *xx* will be replaced by 01 or 02 if FROMADDR or TOADDR, respectively, is not the label of a location of storage; by 03 if OUTPUTUNIT is not the label of one of the acceptable units listed under "Source Program Format." A NOP will be generated.

SNAPS COUNTER ZERO

The SNAPS counter is zero, rendering the SNAP statement ineffective. A NOP will be generated; this will at once be an aid to patching and a literal implementation of the source statement as written.

TOO MANY PARAMETERS. WILL IGNORE

More than six entries have been written into the operand portion of a SNAP statement. The *first* six will be accepted and coding generated accordingly.

If an error is encountered when attempting to write a record on tape, the tape is backspaced and the entire output area of 16 words is typed. Erroneous records will appear on the typewriter only and will not be duplicated on tape. If an error is encountered during a Unit Record Write, the erroneous record is typed as well as printed.

Examples

Each use of SNAP causes the generation of a calling sequence of four words. Any of its components may be addressed through reference to ANYLABEL with appropriate address adjustment and field definition. A closed subroutine is generated out-of-line once per segment. This routine, which is not shown here, occupies about 140 locations and performs the editing to create output records, issues the write commands, and controls the number of records written. On completion of the print-out, the routine returns to the source program through an unconditional Branch to 0003 plus Index Word 94. The format of the calling sequence is as follows:

Line (3–5)	Label (6–15)	Operation (16–20)	OPERAND (21–45)
01	ANYLABEL	BLX	94,,SNAP001,.A
02		DRDW	+FROMADDR,,TOADDR
03		UW	OUTPUTUNIT,,SNAP018,.A
04		DC	
05			+XXXXXXXXXX
06			

The operand of the third instruction, UW, could also be PTW. The constant on line 05 is composed of the following:

Digit Positions	*Contents*
0, 1	The counter, SKIPSPRIOR
4, 5	The counter, SNAPS
8, 9	The counter, SKIPSAFTER
7	1, if SKIPSAFTER is blank

The following is an example of acceptable coding for the SNAP macro-instruction. The associated source-program entries are given, followed by the SNAP statement, coding generated in-line, and coding generated out-of-line.

```
PAGE AA    PROGRAM                                              7070 COMPILER SYSTEM VERSION OMY08, CHANGE LEVEL 00001.  PAGE AA

LN CDREF  LABEL      OP     OPERAND                                                     CDNO FD  LOC  INSTRUCTION     REF

01  891   *                 SNAP EXAMPLE 1
02  892   SNAPOUTP   EQU    19,CU
03  893   *
04  894   ANYLABEL   SNAP   0,99,SNAPOUTP,1,2,3
05       X ANYLABEL  BLX    94,SNAP001.A                                                00001    0325 +0200940357
06       X           DRDW   +0,99                                                                0326 +0000000099
07       X           PTW    SNAPOUTP,SNAP018.A                                                   0327 +8100930356
08       X           DC                                                                               +0003280328
09       X                  +0100020003                                                       09 0328 +0100020003     0328
10  895   *
11  896   *      THE FOLLOWING IS GENERATED OUT OF LINE
12  897   *
13       X SNAP002.A DA     1                                                                         +0003290333
14       X                  49,49                                                             99 0333                 0333
15       X           DA     1                                                                         +0003340349
16       X SNAP015.A        00,159                                                            09 0334                 0334
17       X SNAP016.A DRDW   +SNAP015.A+2,SNAP015.A+3                                    00002    0350 +0003360337
18       X           DRDW   +SNAP015.A+5,SNAP015.A+6                                             0351 +0003390340
19       X           DRDW   +SNAP015.A+8,SNAP015.A+9                                             0352 +0003420343
20       X           DRDW   +SNAP015.A+11,SNAP015.A+12                                           0353 +0003450346
21       X           DRDW   -SNAP015.A+14,SNAP015.A+15                                           0354 -0003480349
22       X SNAP017.A DRDW   -SNAP015.A+2,SNAP015.A+13                                   00003    0355 -0003360347
23       X SNAP018.A DRDW   -SNAP015.A,SNAP015.A+15                                              0356 -0003340349
24       X SNAP001.A PC     +1111111111                                                          0357 +5500000470
25       X           ZST1   SNAP002.A                                                            0358 -1100090329

24       X SNAP020.A ZA1    2(8,9)+X94 LOAD SKIP CTR                                             0453 +1394890002
25       X           STD1   2(0,1)+X94                                                           0454 -1294010002
26       X           ZA1    2(4,5)+X94 LOAD SNAP CTR                                    00023    0455 +1394450002
27       X           BZ1    SNAP021.A+2 MAKE NOP IF SNAPCTR ZERO                                 0456 +1000090460
28       X           B      SNAP001.A+4                                                          0457 +0100090361
29       X SNAP021.A ZA1    2(7,7)+X94 SW7 TEST FOR SKIP                                         0458 +1394770002
30       X           BZ1    SNAP020.A SETUP SKIPSAFTER CTR                                       0459 +1000090453
31       X           ZA1    -0100000000 SETUP SKIPSAFTER FOREVER                        00024    0460 +1300090471
32       X           XS     94,1                                                                 0461 -4700940001
33       X           ZST1   0+X94                                                                0462 -1194090000
34       X           ZST1   1+X94                                                                0463 -1194090001
35       X           ZST1   2+X94                                                                0464 -1194090002
36       X           ZST1   3+X94                                                       00025    0465 -1194090003
37       X SNAP022.A B      SNAP009.A                                                            0466 +0100090440
                     LITERALS
38       X                  +0000000000                                                       09 0467 +0000000000     0467
39       X                  +0000000006                                                       09 0468 +0000000006     0468
40       X                  +0000000012                                                       09 0469 +0000000012     0469
41       X                  +1111111111                                                 00026 09 0470 +1111111111     0470
42       X                  -0100000000                                                       09 0471 -0100000000     0471
43       X                  'SNAP '                                                     00027 09 0472 '8275617700     0472
44       X                  '+'                                                               01 0473 '20             0473
45       X                  '-'                                                               23 0473 '  30           0473
46       X                  'A'                                                               45 0473 '    61         0473
```

SNAP Example 1

A listing of the index word area of storage is desired on tape for two consecutive passes of the program through this area following the first pass through. Tape listings are then omitted for the next three passes. Control then alternates between producing the two tape listings and causing the three skips for the duration of the program.

The Processor

The Organization of the Processor

The Autocoder processor performs functions such as assembly, compilation, and generation for the Autocoder portion of the IBM 7070/7074 Compiler Systems Tape. In addition to Autocoder, the Compiler Systems Tape contains the compilers for FORTRAN, the Report Program Generator, and Commercial Translator. The runs which perform these various functions for all of these compilers fall into two general categories: Compile Runs and Generator Runs. These runs are described in the 7070/7074 Data Processing System Bulletin "IBM 7070/7074 Compiler Systems: Operating Procedure," form J28-6105.

Compilation of a source-language program is performed by a *Compile Run*. The program is converted to machine language and shown on the output listing along with the original symbolic instructions, and then punched into a condensed program deck. A *Generator Run* compiles input statements which create a new macro generator and, if desired, enters the resultant program on a new systems tape. In addition, changes to the Compiler Systems Tape are processed during a *Systems Run* and an updated systems tape is produced.

The operation which is used most frequently in Autocoder processing is that of compilation; this operation is performed by a Compile Run, as well as the compilation portion of a Generator Run. The function of the processor during compilation is discussed on the following pages.

The Autocoder processor consists of the following major sections:

Systems Control: This section of the processor exercises overall control over the compilation process and is principally charged with "housekeeping" functions.

Phase I: This is the first phase in the conversion of the source program to an object program. Phase I scans the input statements of the source program and creates records which will constitute input to Phases II and III.

Phase II: This is the second phase in the conversion of the source program to an object program; Phase II will not become active until Phase I has completed its role in processing. Phase II, in conjunction with the macro generators, compiles the macro-instructions of a program.

Phase III: This is the third and last phase in the conversion of the source program to an object program; Phase III will not become active until Phase II has completed its role in processing. Phase III assembles the final program on the basis of records received from Phases I and II. The output from Phase III consists of a machine-language program and a program listing.

Autosort: Autosort is the portion of the processor which contains appropriate sort routines which are called when it is necessary for the processor to reorder its records for a pass.

Macro Generators: The macro generators and function subroutines are contained on the Compiler Systems Tape and are made available when required to compile macro-instructions in Phase II.

The functions of Systems Control and Phases I, II, and III during compilation and assembly will now be described in more detail.

Systems Control

The chief responsibilities of Systems Control during compilation are as follows:

1. To determine the type of run to be made.
2. To check the validity of the operating options.
3. To supervise the assignment and readiness of the various tape units.
4. To locate and load the various coding blocks on the systems tape.
5. To turn control over to the coding blocks at appropriate times to effect actual processing.

In addition to these functions, Systems Control incorporates the desired modifications, additions, or deletions to the system on a new Compiler Systems Tape when required by a Systems Run or a Generator Run.

The Systems Control program consists of ten sections, called Communication Record, Systems Control 1-8 (SYCL1, SYCL2, etc.), and Update.

Communication Record

The Communication Record consists entirely of data; it contains no instructions to be executed. The data includes the characteristics of the processor machine, the object machine, and the object program, as specified by the programmer through the use of Operating Option Control Cards. The Communication Record also includes various codes generated within the processor for communication between coding blocks.

Systems Control 1 (SYCL1)

SYCL1 loads the Communication Record and turns control over to SYCL3. This section of Systems Control also contains the input/output routines and the Systems Tape Control; both of these remain in storage throughout the compilation process. The Systems Tape Control locates, loads, and transfers control to the various coding blocks.

Systems Control 2 (SYCL2)

SYCL2 is generally called only at the end of a Compile Run to identify halts and output tapes by messages on the console typewriter. In Multifile Runs, it tests for the presence of another program to be compiled and turns control over to SYCL3 again if indicated. SYCL2 may also be used to discontinue a run when the processor determines that this is necessary.

Systems Control 3 (SYCL3)

SYCL3, the largest and most complex portion of Systems Control, initiates the desired run. It reads the various options into the Communication Record, does some validity checking of options, and may type error messages and even cause a halt. It opens the required input and work tapes and does whatever else is necessary in the way of "housekeeping" functions to prepare the type of run specified by the RUN Control Card. If this is an Autocoder Compile Run, Phase I is called.

Systems Control 4-8 (SYCL4-SYCL8)

These sections, which provide for the mounting and dismounting of tapes, are always called by, and in turn will call, a routine other than a Systems Control routine. No two of these routines are identical, but they are quite similar and are included on the systems tape as one program section at five different points. Of these programs, only SYCL6 is loaded during Compile Runs.

Update

Update is a program controlling changes to the systems tape; this program is not loaded during a Compile Run.

Phase I

Phase I reads the source program from cards or tape. Each input statement is assigned a serial number, for internal use by the processor, to govern program order during assembly. Source program page and line numbers are retained and will reappear in the program listing, but they are not utilized during the assembly process. A program identification record is created from columns 76-80 of the first program card. This identification applies to the entire program to be compiled, and columns 76-80 of all subsequent cards are ignored.

As the input statements are scanned by Phase I, a certain amount of error checking is done, and, where necessary, error messages are issued which will ultimately appear in the program listing. In general, Phase I confines its validity checking to such major format violations as omission of operation codes or operands, malformation of the operand portions of macro-instructions, use of illegal characters, errors in field definition, use of unallowable address types, exceeding machine capacity, or exceeding the maximum permissible number of LITORIGIN statements. If Phase I encounters an input statement which cannot be processed, it will usually retain as much of the statement that was scanned, or it will generate a NOP. An error message will be issued.

The chief function of Phase I is to furnish records to Phases II and III that will enable them to compile the finished program; this is done by analyzing the source program statements and by writing records for the label and operand entries. These "element" records contain pertinent information about the source program entry, such as its serial number, its operation code, and other attributes relevant to the processor. Output from Phase I consists of two files, File A and File B. File A, which contains all of the element records, goes directly to Phase III. File B, which contains all records that may be required for the compilation of macro-instructions, becomes input to Phase II. (Phase II will receive not only records derived from macro-instructions themselves, but also label records from all other instructions having symbolic labels, since these may be referenced by macro-instruction operands.)

Input statements are scanned one at a time. When comments cards are encountered, appropriate records are written onto File A for inclusion in the final program listing. Other source statements are classified according to their operation code, which will be that of a declarative statement, a control statement, an imperative statement (either a symbolic machine instruction or a macro-instruction), or blank. If the operation code does not fall into any of these categories, or if a blank operation code occurs on a card following one that does not permit continuations or subsequent entries, the code is converted to a NOP, and an error message will appear for that line of the program listing.

Phase I processes its input statements in three successive passes, each of which is processed by a separate coding block. These coding blocks contain the programs that will process statements with the following operation codes, respectively:

Pass 1 *First Coding Block*	*Pass 2* *Second Coding Block*	*Pass 3* *Third Coding Block*
Symbolic machine instructions	DA	XRESERVE Control
	CODE	SRESERVE Control
DRDW	DC	XRELEASE Control
EQU	DLINE	SRELEASE Control
BRANCH Control	DSW	DTF
END Control	ORIGIN Control	DUF
	LITORIGIN Control	Macro-instructions

The first coding block is loaded into storage and the entire source program is scanned for the statements whose programs are contained on this block. These statements are processed at once; the others are written onto a temporary work tape. The second coding block is then loaded into storage. The work tape containing the statements which have not been processed is scanned. The statements whose programs are contained on the second coding block are processed; the remaining statements are again written onto a work tape. Finally, the third coding block is loaded into storage and the remaining statements are processed. Thus, each of the three coding blocks needs to be loaded into storage but once.

This procedure, of course, temporarily destroys source program (i.e., page and line) order, but this order will ultimately be restored by means of the internal serial number assigned to each statement. In addition, input statements are generally taken apart, with separate records being written for the label and the one or more operand entries. These individual components of a given source statement are also numbered and provided with indicators that will permit eventual recomposition of the source statements.

Phase II

Phase II processes the records on File B in order to perform the compilation of macro-instructions. Its output, a series of records corresponding to symbolic machine instructions and declarative statements, is written on File A.

The first step of Phase II is a sort of File B, which contains all label records of the declarative or imperative statements having symbolic labels; the records are grouped by symbolic name.

The sort is followed by an Information Transfer pass. This pass provides additional necessary information concerning the characteristics of fields occurring in the operands of macro-instructions; this is performed by transferring the characteristics shown in the label records into the macro-instruction operand records.

At the conclusion of the Information Transfer pass, all records other than macro-instruction labels and operands are dropped from File B. The remaining records are sorted and grouped by macro-instruction operation code. Within this grouping, the records are arranged according to their order of appearance in the program. This sorting avoids frequent reloading of the same macro generator since the processor will be able to operate successively on all appearances of each macro-instruction.

The generating portion of Phase II is governed by a program called Phase II Control. This program determines which macro generator is required and calls the appropriate coding block from the library. While one generator is being executed, Systems Tape Control (see "Systems Control 1 (SYCL1)") positions the tape in anticipation of the next required generator. Phase II Control sets up certain counters required by the generators, such as parameter counters and counters to record the number of previously generated labels. In addition, it causes the macro generator required to be loaded into storage, furnishes appropriate "parameter" records to it as a basis for analysis, and then turns control over to the generator itself.

The parameter records are not quite identical to the element records written by Phase I and completed in the Information Transfer pass, but they are based on them. The parameter records need not contain the operation code or the serial number; this information is stripped from the element record and temporarily stored elsewhere, to be attached again to output records. In general, the rest of the element record goes into the parameter record area unchanged, but some items that cannot be conveniently accommodated there, such as long alpha-

meric literals, the input texts of ARITH and LOGIC statements, etc., are stored in a separate area. RDWS defining their location are made available to the generator in the parameter record area. Records of address adjustment and indexing are not preserved as independent entities; the pertinent information is entered into the records of the parameters they modify. Since one macro-instruction is compiled at a time, the parameter record area will, at any one time, contain only those records derived from a single macro-instruction.

Generated output is not, strictly speaking, produced by the macro generators themselves, but by the GENER subroutine, which is in storage during the entire generating portion of Phase II. The macro generators contain "model statements" which, together with the parameter records and certain indicators set by the generator, serve as a guide for the construction of appropriate records by GENER. These records, which are written onto File D, correspond to a sequence of symbolic machine instructions or macro-instructions that will accomplish the operations indicated by the original macro statement.

Certain Autocoder macro generators pass on portions of their work to other "lower-level" macro-instructions. For example, the LOGIC macro-instruction often passes work on to the COMP macro-instruction. In such cases, the generated output on File D will include both macro-instructions which require further compilation and symbolic machine instructions which do not. Therefore, after the generating portion of Phase II, the temporary output file, File D, is edited into two further files. Symbolic machine instructions, declarative statements, and control statements are written onto File A as eventual input to Phase III; an editing program changes the format of these records into a format identical to that produced by Phase I. The generated macro-instructions which require further compilation are edited to a format identical to that produced by Information Transfer. Since these records have already been augmented to include data characteristics, the records are re-entered at a point *following* the Information Transfer pass and *beginning* with the sort by macro-instruction operation code.

When no further macro-instructions remain to be compiled, all Phase II output will have been edited and written onto File A; this file then goes to Phase III for assembly.

Phase III

At the start of Phase III all macro input statements have been reduced to sets of elementary Autocoder statements consisting of machine instructions, declaratives, and several types of comments records. The input to Phase III was written on File A by Phases I and II. After sorting these statements back into the order of the source program, the statements are assigned machine locations, the operands are replaced by their equivalent machine location values, and the individual machine instructions are built. These are shown on the output listing, along with the original symbolic instructions, and then packed into a condensed program deck.

Phase III is organized into six passes and four sorts, as follows:

1. Record Construction
2. Sort Serial-Request File
3. Serial Transfer
4. Sort Statement File
5. Assignment
6. Sort Symbol File

7. Information Transfer
8. Sort Operand File
9. Output
10. Message

The Assignment pass is the central processing section of Phase III. Its primary function is to assign a machine location to each statement as it is encountered. The passes and sorts preceding the Assignment pass prepare the input for this pass. The succeeding passes and sorts handle the final preparation and editing of the output.

Record Construction

The first pass, the Record Construction pass, writes two types of records. On one file, statement records are constructed from the element records written by Phases I and II. A statement record is written for each statement; it contains the label, operation code, one or two operands, field control, address adjustment, indexing, and remarks. On a second file, Record Construction creates the following request records:

1. A Serial Record for each labeled statement (including EQU statements).
2. An Equate Request Record for each symbolic operand of each EQU statement.
3. An Index Assignment Request Record for each symbolic index word appearing either in the operand of an index word operation such as XL or XZA, or as indexing.
4. A Switch Assignment Request Record for each symbolic electronic switch appearing in the operand of an electronic switch operation such as BES or ESN.
5. An Implicit Indexing Request Record for each labeled DA subsequent entry included under a DA header line showing implicit indexing.
6. An Origin Request Record for each symbolic NAMEONE operand of an ORIGIN or LITORIGIN statement.
7. A Literal Request Record for each literal or adcon appearing in the operand of an instruction.

Sort Serial-Request File

The Serial Record and request records created by the Record Construction pass are sorted by symbolic name and, further, by the type code of each request record. This sort places the file in order so that all request records for each symbolic name are grouped together following the Serial Record, if one has been created. The literals and adcons will be sorted into two groups following the request records. These records will be in order by adcon name or by literal value.

Serial Transfer

The Serial Transfer pass selects the request records to be added to the statement file created during the Record Construction pass; this statement file will be processed during the Assignment pass.

Each Index Assignment Request Record and each Switch Assignment Request Record is read; the request record for each index word and electronic switch with the lowest card serial number (i.e., the record of the *first* occurrence of the symbolic name in the program) is passed on to the statement file.

Serial Transfer retains a record of the card serial number of each Index Assignment Request Record it passes to the statement file. This number is transferred to the Implicit Indexing Request Record for each labeled DA subsequent entry. These records are passed on to the statement file.

Equate Request Records have been created in the Record Construction pass for the symbolic operands of EQU statements. In addition, Serial Records have been created for the label of each EQU statement. The Serial Transfer pass matches the Serial Records with their corresponding Equate Request Records. The card serial number from the Serial Record is added to the Equate Request Record; the Equate Request Records are then passed on to the statement file.

Finally, the Serial Transfer pass processes the Literal Request Records. This pass essentially creates a DC for the complete set of literals and adcons. A DC subsequent entry, with either the literal value or the adcon as its operand, is created for each unique literal or adcon; duplicate references are read and dropped.

Sort Statement File

The complete statement file is now sorted by card serial number to give the Assignment pass a file in source program order. The request records have all been given card serial numbers and will be merged with their respective statements. Literals will have been numbered so that they fall last in each litorigin segment of the program.

Assignment

The Assignment pass assigns a machine location to each statement as it is encountered. A statement might be a symbolic machine instruction which occupies one word of storage or it might be a declarative which occupies any amount of storage from part of one word to many words. The Assignment pass assigns a reference address to each statement regardless of the type; the reference address is composed of the address and field control of the part of the statement that occupies the first word of storage assigned to this statement.

In addition to its function of assigning a reference address to each statement, the Assignment pass creates two types of records, Definition Records and Operand Records, which are placed in the Information Transfer file. Definition Records are created for each symbolic label encountered; the record contains the reference address which has been assigned to the labeled statement. Operand Records are created for each symbolic operand encountered.

Symbolic index words and electronic switches create special cases; these symbols must be assigned machine addresses. The statement file sort has placed an Index Assignment Request Record following the instruction containing the first reference to the symbolic index word; likewise, it has placed a Switch Assignment Request Record following the instruction containing the first reference to the symbolic electronic switch. These request records initiate the assignment of a machine location and the creation of a Definition Record to be placed in the Information Transfer file.

Labels of DA subsequent entries whose DA header lines show implicit indexing also create special cases. Implicit Indexing Request Records, which were created by the Record Construction pass, have been placed following the appropriate Index Assignment Request Records. These request records initiate the creation of Definition Records which contain the implicit index value; the Definition Records are placed in the Information Transfer file.

Another special case is equating one symbol to another symbol. An Equate Request Record has been placed following the statement with the same label as the EQU statement operand. The request record initiates the creation of a Definition Record to be placed in the Information Transfer file.

When ORIGIN and LITORIGIN statements are encountered, the Assignment pass must be able to change the address assignment to continue from some previously

defined address. An Origin Request Record for the operand NAMEONE will follow the statement with the same label as the NAMEONE. It will cause an entry in a table containing values of addresses for reference by ORIGIN and LITORIGIN statements.

Sort Symbol File

The Information Transfer file is sorted by symbolic name. Within each set of records for one symbol, Definition Records precede Operand Records.

Information Transfer

The Information Transfer pass will transfer the assigned value in each Definition Record to each operand that refers to the defined symbol. Simultaneously, the pass will produce the Cross Reference Listing which will be written on the end of the statement file.

Sort Operand File

The Operand Records, with assigned values, will be sorted on the page and line number of the statement which contained the symbolic operand; the Operand Records will then be used as input to the Output pass.

Output

The Output pass will read each statement record and reconstruct the card image of the statement, i.e., label, operation code, and operand. The Operand Records that pertain to a statement will be read and the values substituted for symbols in the statement. The pass will construct the machine instructions which will be shown with the original symbolic instructions on the output listing; the machine instructions will be condensed into the output program deck. Any halt instructions that occur will be copied onto the Message file, which will also receive any Message Records that have been added to the statement file during the entire process. After the last statement has been processed, the Message pass will be called.

Message

This pass will first write the memory map, the index word and electronic switch availability tables, and the Cross Reference Listing from the end of the statement file onto the output listing. Next, a component listing from the end of the Operand Record file will be added to the output listing. The list of halt statements are then written, followed by the list of error messages.

Output Listings

Program Listing

The Program Listing is prepared during the Output pass of Phase III. This listing contains the original symbolic instructions with the assembled machine instructions. A sample Program Listing is shown on the opposite page. The heading **(1)** of the Program Listing is "7070 COMPILER SYSTEM VERSION XXXXX, CHANGE LEVEL YYYYY." Appropriate substitutions for the version number and change level number are determined by the identification on the Compiler Systems Tape. A page number **(2)** appears twice on each page. The page numbers are assigned to each page of the listing by Phase III; the pages are numbered sequentially by a two-letter symbol (AA, AB, ... through ZZ). The PROGRAM ZZZZZ **(3)** entry is the program identification record; if no identification is used, this entry will be blank.

The listing on the left-hand portion of the page contains the original source-program entries as well as the symbolic machine instructions, DRDWS, literals, etc., that have been generated. The line number **(4)** is assigned by Phase III to each entry. The card reference number **(5)** is the page and line number

```
PAGE AA    PROGRAM  ZZZZZ                7070 COMPILER SYSTEM VERSION XXXXX, CHANGE LEVEL YYYYY.  PAGE AA

LN CDREF   LABEL      OP    OPERAND                              CDNO  FD  LOC   INSTRUCTION   RI REF
01  655    *                ZERO EXAMPLE 5
02  656    DRDWNAME   DRDW  AREANAME
03       X DRDWNAME   DRDW  +AREANAME,AREANAME+24                00001     0325 +0003270351    11
04       X            DRDW  -AREANAME+25,AREANAME+49                       0326 -0003520376    11
05  657    *
06  658    AREANAME   DA    2,,0+INDEXWORD                                      +0003270376
07  659    FIELDA           00,99                                      09  0327                   0000
08  660    FIELDB           100,103A                                   03  0337                   0010
09  661    FIELDC           104,129A                                   49  0337                   0010
10  662    FIELDD           130,249A                                   09  0340                   0013
11  663    *
12  664    ANYLABEL   ZERO  DRDWNAME                                                                  M
13       X ANYLABEL   ZA2   +0                                   00002     0377 +2300000379    01
14       X            STD2  DRDWNAME(0,9)                                  0378 -2200090325    01
15  665    *
16  666    *       THE FOLLOWING IS GENERATED OUT OF LINE
17  667    *
                      LITERALS
18       X                  +0                                     00  0379 +0                 00 0379
```

assigned by the programmer to each source-program entry. The label **(6)**, operation **(7)**, and operand **(8)** columns contain the source-program entries and the generated statements. Any remarks originally contained in the operand of the input statements are printed on the listing. An X **(9)** preceding the label column of an entry indicates that the instruction has been generated. An apostrophe **(10)** appears in the listings throughout this manual and indicates the use of an @ character; the ' is equivalent to the @ character on the H type wheel configuration.

The listing on the right-hand portion of the page contains the assembled machine instructions. The card number **(11)** is assigned by the Output pass of Phase III to each condensed card punched out. When the source statement specifies field definers for less than the full word, the field definition **(12)** is indicated. The location **(13)** is the machine location assigned to the instruction. The instruction **(14)** is the actual assembled machine instruction. The relocation indicators **(15)** are two-digit numbers assigned each assembled instruction to indicate which part(s) of the instruction must be adjusted if relocation of all or part of the program is desired. The reference address **(16)** is either (a) the actual address assigned to literals or to DA, DC, or DLINE subsequent entries, or (b) the relative address of DA subsequent entries when a relative address is specified in the DA header line. An M **(17)** following the reference address column indicates that an error or warning message is associated with that line.

Origin Counter Listing

An Origin Counter Listing follows the Program Listing. Each counter is listed in alphameric order; the initial value, last value, highest value, and lowest value for each counter follow. An example of an Origin Counter Listing appears on page 94.

Availability Table

An Availability Table follows the Origin Counter Listing. This table first lists the electronic switches which remain available for assignment; the index words which are available for assignment follow. If no index words or if no electronic switches remain, the word "NONE" appears in place of the list of switches or index words.

Cross Reference Listing

The Cross Reference Listing is prepared by the Information Transfer pass of Phase III. All symbolic labels, symbolic index words, symbolic switches, CODE fields, literals, and adcons are listed in alphameric order. Each of these entries is followed first by page, line, and actual location of all operand usages of that label.

Component Assignment Listing

The Component Assignment Listing presents all of the electronic switch and index words used in the compiled program. The symbolic name(s), if any, assigned to each of these components is included in this listing.

Halt Listing

A Halt Listing follows the Component Assignment Listing. Each halt instruction occurring in the compiled program will be listed in the order in which it occurs in the program. The halt instructions in the listing will appear in the same format as they appear in the Program Listing.

Message Listing

The Message Listing is provided by the Message pass of Phase III; this listing immediately follows the Halt Listing. The actual message is printed, as well as the page and line number (referred to by **(2)** and **(4)** on the Program Listing) of the entry concerned.

Appendix A: Relationship of 7070/7074 Autocoder to Basic Autocoder and Four-Tape Autocoder

The advanced programming capabilities of the 7070/7074 Autocoder system, as compared to the 7070/7074 Four-Tape Autocoder, are due to more powerful macro-instructions, extensive control operations over processing, and increased input/output options. To effect these improvements, distinct statement types and language specifications peculiar to Autocoder have been developed. However, as in Four-Tape Autocoder, the new language characteristics are additions rather than changes to the 7070/7074 Basic Autocoder language. It is therefore possible for either of the more powerful systems to process any program that can be assembled with Basic Autocoder without modification. The converse is untrue since Basic Autocoder is not designed to process the advanced programming features provided in the larger systems. Similarly, since Autocoder and Four-Tape Autocoder each use a unique type of macro-instructions and other programming functions, neither is designed to process *all* programs which can be assembled by the other.

If desired, the user may advance from Four-Tape Autocoder to the Autocoder system by using macro generators to duplicate the substitution-type macros used in a program, or by recoding the program so that macro-instructions provided with or added to the Autocoder system may be utilized. In order for a program coded in Four-Tape Autocoder to be fully compatible with the programming requirements of the Autocoder processor, certain additional changes in the coding and treatment of some programming functions must be made. These differences are outlined in Appendix B.

Appendix B: Differences Among 7070/7074 Autocoder Systems

The areas in which 7070/7074 Autocoder differs from 7070/7074 Basic Autocoder or 7070/7074 Four-Tape Autocoder are outlined below.

Coding Sheet, Operand

Basic Autocoder. Columns 21-60.

Four-Tape Autocoder and Autocoder. Columns 21-75.

Address Types

A blank address in a LITORIGIN statement has the following significance:

Basic Autocoder. Not to be used.

Four-Tape Autocoder and Autocoder. Assignment continues at one beyond the highest location previously assigned, except for the locations assigned by the special "s" counter.

The number of symbolic labels that can be used in a source program is as follows:

Basic Autocoder. The number is limited by size of storage area available for the symbol table.

Four-Tape Autocoder. The symbol table is written as a block on tape when the area in storage is filled. The number is limited only by the number of blocks that can be written on the symbol tape.

Autocoder. There is no practical limit to the number of symbolic labels that can be used in a source program.

Index Words

Basic Autocoder and Four-Tape Autocoder. Once reserved, index words cannot be made available for reassignment later in the same program.

Autocoder. Index words may be reserved by means of an XRESERVE statement. Index words which have been previously assigned by *any* method may be made available for later assignment by means of an XRELEASE statement.

Electronic Switches

Basic Autocoder and Four-Tape Autocoder. Once reserved, electronic switches cannot be made available for reassignment later in the same program.

Autocoder. Electronic switches may be reserved by means of an SRESERVE statement. Electronic switches which have been previously assigned by *any* method may be made available for later assignment by means of an SRELEASE statement.

DA—Define Area

A DA header line has the following differences:

Basic Autocoder and Four-Tape Autocoder. N may be from 1 to 999. The number of storage words that may be reserved for an area is a maximum of 999.

Autocoder. N may be from 1 to 9999. The number of words that may be reserved is limited only by the size of the object machine. In addition, a symbolic or actual index word may be appended to the relative address to facilitate the writing of indexed instructions which reference fields in the defined area.

The subsequent entries have the following differences:

Basic Autocoder and Four-Tape Autocoder. These entries indicate the names of fields and the position of the field within the area only. A CODE entry is not permitted.

Autocoder. The format or characteristics of a field, i.e., numerical and an automatic-decimal number, numerical and a floating-decimal number, etc., may also be included following the indication of the position of the field within the area. A CODE entry may be used.

DC — Define Constant

Basic Autocoder and Four-Tape Autocoder. Numerical constants may contain a maximum of ten digits. In alphameric constants, the @ symbol may appear only if it immediately precedes the terminal @ symbol. The @ symbol may appear in remarks which are on the same line as an alphameric constant. An adcon cannot be modified by address adjustment.

Autocoder. Numerical constants may be in standard floating-decimal format or in the form of automatic-decimal numbers of up to twenty digits in length. The @ symbol may appear *anywhere* within an alphameric constant. The @ symbol *cannot* be used in remarks which are on the same line as an alphameric constant.

EQU — Equate

Basic Autocoder and Four-Tape Autocoder. The symbolic address to which a symbolic *name* is being equated must have appeared as a label *earlier* in the sequence of program entries. A symbolic name cannot be equated to a digit value.

Autocoder. A symbolic name may be made equivalent to a symbolic address which appears as the label for an entry anywhere in the source program. A symbolic name can be equated to a digit value.

Origin Control

Basic Autocoder and Four-Tape Autocoder. The location assignment counter used by the processor may be set to the starting location specified in the operand of the ORIGIN statement. A blank operand signifies that the contents of the high assignment counter plus 1 is to be used by the processor. The letter "s" following an address in the operand orders the processor not to alter the high assignment counter during the processing of succeeding entries. If an ORIGIN statement does not appear before a location is assigned to the first source-language input entry, the processor will begin the assignment of storage locations at 0325.

Autocoder. In addition to the automatic location assignment counter, over 250 separate symbolic-location counters may be named by the programmer and used by the processor to control the placement of a program in storage. A blank operand in an ORIGIN statement indicates that the maximum value attained by *any* location counter, other than counter "s", is to be used for the assignment of subsequent locations. The assignment of storage locations will begin at an address specified in the Compiler Systems Tape if an ORIGIN entry does not appear before a location is assigned to the first source-language input entry in a program.

Litorigin Control

Basic Autocoder. Normally, literals will be stored in locations immediately following the highest location assigned to the source program. One LITORIGIN statement, placed at the end of the source program deck or immediately preceding the END Control Card (if used) may be utilized to start the assignment of literals at the location specified in the operand rather than following the highest location used in the program.

Four-Tape Autocoder. Library subroutines, as well as literals, may be stored in locations immediately following the highest location assigned to the source program. However, the LITORIGIN statement may be used to insert these generated subroutines into the program at the point where the LITORIGIN entry appears. More than one such entry may be used in a program but each must be

followed by an ORIGIN statement unless the LITORIGIN statement is either the last entry or is followed by an END Control operation.

Autocoder. Same format as Autocoder ORIGIN statement. May be used to regulate the placement of materials generated out-of-line, i.e., implicit adcons, area definitions, etc., as well as literals. Up to 25 LITORIGIN statements may be used in a program to cause the assignment of locations to all material generated up to the point at which another LITORIGIN entry is encountered. Each phase of a multi-phase program may thus be loaded with its own literals, generated constants, etc. ORIGIN statements are not required following a LITORIGIN entry.

End Control

Basic Autocoder and Four-Tape Autocoder. If an END statement is not used, the processor will generate an unconditional branch to location 0325.

Autocoder. If an END statement is not used, or if used with a blank operand, the processor will generate an unconditional branch to the address specified in the Compiler Systems Tape.

Macro-Instructions

Basic Autocoder. Cannot process macro-instructions or DTFS per se.

Four-Tape Autocoder. The 7070 Input/Output Control System macro-instructions may be used with the following restrictions:

1. When using PUT macro-instructions with Four-Tape Autocoder, the name preceding IN must be the name of either an RDW which defines one area or a tape input file. The use of a field name is not allowed unless the field name is the label of an RDW that defines the field.
2. The name of a card input file may not be used in the PUT macro-instruction if the output file is a tape file.
3. A record from a card input file may be included in a tape output file if the unit record area is defined by one RDW; the PUT would then be written using the name of the RDW; i.e., the name preceding the word IN would be the same as the fourth item in the DUF entry of the card input file.
4. Only nine tape files may be named in the operand of an OPEN or CLOSE macro.
5. The symbolic names IOCSIXF, IOCSIXG, and IOCSIXH may not appear in the operand of the DIOCS statement.
6. Any index words to be used in the DIOCS statement must be actual. These may later be equated to a symbolic name.
7. Comments cards cannot be inserted between DTFS or among DTF subsequent entries.

Other macro-instructions that can be used are substitution type—values in the operand of the macro in the source program are used to complete the labels, operation codes, etc. of a sequence of instructions contained in the Library portion of the Systems Tape.

Autocoder. The 7070 Input/Output Control System macro-instructions can be used without the restrictions noted above. The additional macros that can be used with Autocoder are processed by macro generators and therefore differ from the substitution type of macro-instructions that can be used with Four-Tape Autocoder.

Appendix C: Reserved Index Words

The Compiler Systems Tape will not allow assignment of *symbolic* names to certain index words. The address and special function of each of these index words are listed below.

Address	*Function*
0093	Used by Autocoder and FORTRAN floating-decimal subroutines.
0094	Used by Autocoder and FORTRAN normal subroutines.
0095	Used by SPOOL operations on Channel 2.
0096	Used by SPOOL operations on Channel 1.
0097	Priority address word.
0098	Table lookup indexing value and found address.
0099	Address of priority final status word.

Appendix D: 7070/7074 Operation Codes by Autocoder Mnemonics

In the following list, the symbols in the operand column indicate what is permissible in the operand and the order in which this information must be written on the coding form for each symbolic machine instruction.

In all cases where an "A" has been indicated, a literal may also be used. The list, however, indicates a literal, "L," and also field definition, "F," only where it would seem to be of practical value. Caution is advised when using literals with operation codes which do not specifically indicate them.

Operand Symbol Key

Symbol	*Meaning*	*Type of Coding*	*Range of Actual*
A	Address	Symbolic or actual	Any storage location
AF	Arm and file	Symbolic or actual	00-03, 10-13, 20-23
B	Blank		
C	Channel number	Symbolic or actual	1-4
CU	Channel and unit	Symbolic or actual	10-49
D	Digit	Actual	0-9
F	Field Definition	Actual (Enclosed in parentheses)	0-9
I	Unit record latch	A or 1, B or 2	1-2
L	Literal		
N	Number	Actual	0-10 (for normal shifts) 0-20 (for coupled or split shifts) 0-9999 (for index word codes)
P	Digit position	Actual (Enclosed in parentheses)	0-9 (for CD) 0-19 (for split shifts)
Q	Inquiry synchronizer	Symbolic or actual	1-2
S	Unit record synchronizer	Symbolic or actual	1-4
SN	Alteration switch	Symbolic or actual	1-4
SW	Electronic switch	Symbolic or actual	1-30
X	Index Word	Symbolic or actual	1-99
,	Used as a separator and must be written on the coding sheet (unless the address which follows is blank).		
/	Used to indicate the word "or" (e.g., A/L means either an address or a literal).		
#	Used to indicate an accumulator number (1, 2, or 3, which must appear in place of the # symbol).		

Alphabetic List of 7070/7074 Operation Codes by Autocoder Mnemonics

Mnemonic	*Operation*	*Operand*	
A#	Add to accumulator #	A/L	F
AA	Add absolute to accumulator 1	A/L	F
AAS#	Add to absolute storage from accumulator #	A	F
AS#	Add to storage from accumulator #	A	F
B	Branch	A	
BAL	Branch if any stacking latch is ON	A	
BAS	Branch if alteration switch is ON	SN,A	
BCB	Branch if channel is busy	C,A	
BCX	Branch compared index word	X,A	
BDL	Branch if disk storage latch is ON	AF,A	
BDX	Branch decremented index word	X,A	
BE	Branch if equal	A	
BES	Branch if electronic switch is ON	SW,A	
BFV	Branch if field overflow	A	
BH	Branch if high	A	
BIX	Branch incremented index word	X,A	
BL	Branch if low	A	
BLX	Branch and load location in index word	X,A	
BM#	Branch if minus in accumulator #	A	
BQL	Branch if inquiry latch is ON	Q,A	
BSC	Branch if sign change	A	
BSF	Branch if electronic switch is ON and set OFF if ON	SW,A	
BSN	Branch if electronic switch is ON and set ON if OFF	SW,A	
BTL	Branch if tape latch is ON	CU,A	
BUL	Branch if unit record latch is ON	I,A	
BV#	Branch if overflow in accumulator #	A	
BXM	Branch if index word is minus	X,A	
BXN	Branch if indexing portion in index word is nonzero	X,A	
BZ#	Branch if zero in accumulator #	A	
C#	Compare accumulator # to storage	A/L	F
CA	Compare absolute in accumulator 1 to absolute in storage	A/L	F
CD	Compare storage to digit	A(P),D	
CSA	Compare sign to alpha	A	
CSM	Compare sign to minus	A	
CSP	Compare sign to plus	A	
D	Divide	A/L	F
DAR	Disk storage arm release	AF	
DLF	Disk storage latch set OFF	AF,A/B	
DLN	Disk storage latch set ON	AF,A/B	
DR	Disk storage read	C,A/L	
DW	Disk storage write	C,A/L	
EAN	Edit alphameric to numerical	X,A/L	
ENA	Edit numerical to alphameric	X,A/L	
ENB	Edit numerical to alphameric with blank insertion	X,A/L	
ENS	Edit numerical to alphameric with sign control	X,A/L	
ESF	Electronic switch set OFF	SW,A/B	
ESN	Electronic switch set ON	SW,A/B	
FA	Floating add	A/L	
FAA	Floating add absolute	A/L	
FAD	Floating add double precision	A/L	

Mnemonic	*Operation*	*Operand*	
FADS	Floating add double precision and suppress normalization	A/L	
FBU	Floating branch underflow	A	
FBV	Floating branch overflow	A	
FD	Floating divide	A/L	
FDD	Floating divide double precision	A/L	
FM	Floating multiply	A/L	
FR	Floating round	A/B	
FS	Floating subtract	A/L	
FSA	Floating subtract absolute	A/L	
FZA	Floating zero and add	A/L	
HB	Halt and branch	A	
HMFV	Halt mode for field overflow	A/B	
HMSC	Halt mode for sign change	A/B	
HP	Halt and proceed	A/B	
LE	Lookup equal only	A/L	F
LEH	Lookup equal or high	A/L	F
LL	Lookup lowest	A/L	F
M	Multiply	A/L	F
MSA	Make sign alpha	A	
MSM	Make sign minus	A	
MSP	Make sign plus	A	
NOP	No operation	A/B	
PC	Priority control	A/L	
PDR	Priority disk storage read	C,A/L	
PDS	Priority disk storage seek	A	
PDW	Priority disk storage write	C,A/L	
PR	Priority release	A/B	
PTM	Priority tape mark write	CU	
PTR	Priority tape read	CU,A/L	
PTRR	Priority tape read per record mark control	CU,A/L	
PTSB	Priority tape segment backspace	CU,A/L	
PTSF	Priority tape segment forward space	CU,A/L	
PTSM	Priority tape segment mark write	CU	
PTW	Priority tape write	CU,A/L	
PTWC	Priority tape write with zero elimination and per record mark control combined	CU,A/L	
PTWR	Priority tape write per record mark control	CU,A/L	
PTWZ	Priority tape write with zero elimination	CU,A/L	
QLF	Inquiry latch set OFF	Q,A/B	
QLN	Inquiry latch set ON	Q,A/B	
QR	Inquiry read	Q,A/L	
QW	Inquiry write	Q,A/L	
RG	Record gather	X,A/L	
RS	Record scatter	X,A/L	
S#	Subtract from accumulator #	A/L	F
SA	Subtract absolute from accumulator 1	A/L	F
SL	Shift left coupled	N	
SL#	Shift left accumulator #	N	
SLC	Shift left and count coupled	X	
SLC#	Shift left and count accumulator #	X	
SLS	Shift left split	N(P)	
SMFV	Sense mode for field overflow	A/B	

Mnemonic	*Operation*	*Operand*	
SMSC	Sense mode for sign change	A/B	
SR	Shift right coupled	N	
SR#	Shift right accumulator #	N	
SRR	Shift right and round coupled	N	
SRR#	Shift right and round accumulator #	N	
SRS	Shift right split	N(P)	
SS#	Subtract accumulator # from storage	A	F
ST#	Store accumulator #	A	F
STD#	Store digits from accumulator # and ignore sign	A	F
TEF	Tape end of file turn OFF	CU	
TLF	Tape latch set OFF	CU,A/B	
TLN	Tape latch set ON	CU,A/B	
TM	Tape mark write	CU	
TR	Tape read	CU,A/L	
TRB	Tape record backspace	CU	
TRR	Tape read per record mark control	CU,A/L	
TRU	Tape rewind and unload	CU	
TRW	Tape rewind	CU	
TSB	Tape segment backspace	CU,A/L	
TSEL	Tape select	CU	
TSF	Tape segment forward space	CU,A/L	
TSHD	Tape set high density	CU	
TSK	Tape skip	CU	
TSLD	Tape set low density	CU	
TSM	Tape segment mark write	CU	
TW	Tape write	CU,A/L	
TWC	Tape write with zero elimination and per record mark control combined	CU,A/L	
TWR	Tape write per record mark control	CU,A/L	
TWZ	Tape write with zero elimination	CU,A/L	
TYP	Type	A/L	
ULF	Unit record latch set OFF	I,A/B	
ULN	Unit record latch set ON	I,A/B	
UP	Unit record punch	S,A/L	
UPIV	Unit record punch invalid	S,A/L	
UR	Unit record read	S,A/L	
US	Unit record signal	S,A/B	
UW	Unit record write	S,A/L	
UWIV	Unit record write invalid	S,A/L	
XA	Index word add to indexing portion	X,A/N	
XL	Index word load	X,A/L	
XLIN	Index word load and interchange	X,A/L	
XS	Index word subtract from indexing portion	X,A/N	
XSN	Index word set nonindexing portion	X,A/N	
XU	Index word unload	X,A	
XZA	Index word zero and add to indexing portion	X,A/N	
XZS	Index word zero and subtract from indexing portion	X,A/N	
ZA#	Zero accumulator # and add	A/L	F
ZAA	Zero accumulator 1 and add absolute	A/L	F
ZS#	Zero accumulator # and subtract	A/L	F
ZSA	Zero accumulator 1 and subtract absolute	A/L	F
ZST#	Zero storage and store accumulator #	A	F

List of 7070/7074 Operation Codes by Function

Mnemonic	*Operation*	*Operand*
	TAPE OPERATIONS	
TR	Tape read	CU,A/L
PTR	Priority tape read	CU,A/L
TRR	Tape read per record mark control	CU,A/L
PTRR	Priority tape read per record mark control	CU,A/L
TW	Tape write	CU,A/L
PTW	Priority tape write	CU,A/L
TWR	Tape write per record mark control	CU,A/L
PTWR	Priority tape write per record mark control	CU,A/L
TWZ	Tape write with zero elimination	CU,A/L
PTWZ	Priority tape write with zero elimination	CU,A/L
TWC	Tape write with zero elimination and per record mark control combined	CU,A/L
PTWC	Priority tape write with zero elimination and per record mark control combined	CU,A/L
TSB	Tape segment backspace	CU,A/L
PTSB	Priority tape segment backspace	CU,A/L
TSF	Tape segment forward space	CU,A/L
PTSF	Priority tape segment forward space	CU,A/L
TSEL	Tape select	CU
TRB	Tape record backspace	CU
TRW	Tape rewind	CU
TRU	Tape rewind and unload	CU
PTM	Priority tape mark write	CU
TM	Tape mark write	CU
PTSM	Priority tape segment mark write	CU
TSM	Tape segment mark write	CU
TEF	Tape end of file turn OFF	CU
TSHD	Tape set high density	CU
TSLD	Tape set low density	CU
TSK	Tape skip	CU
TLN	Tape latch set ON	CU,A/B
TLF	Tape latch set OFF	CU,A/B
BTL	Branch if tape latch is ON	CU,A
	DISK STORAGE OPERATIONS	
PDS	Priority disk storage seek	A
DR	Disk storage read	C,A/L
PDR	Priority disk storage read	C,A/L
DW	Disk storage write	C,A/L
PDW	Priority disk storage write	C,A/L
DAR	Disk storage arm release	AF
DLN	Disk storage latch set ON	AF,A/B
DLF	Disk storage latch set OFF	AF,A/B
BDL	Branch if disk storage latch is ON	AF,A
	UNIT RECORD OPERATIONS	
UR	Unit record read	S,A/L
UP	Unit record punch	S,A/L
UW	Unit record write	S,A/L
UPIV	Unit record punch invalid	S,A/L
UWIV	Unit record write invalid	S,A/L

Mnemonic	*Operation*	*Operand*	
US	Unit record signal	S,A/B	
ULN	Unit record latch set ON	I,A/B	
ULF	Unit record latch set OFF	I,A/B	
BUL	Branch if unit record latch is ON	I,A	
	INQUIRY OPERATIONS		
QR	Inquiry read	Q,A/L	
QW	Inquiry write	Q,A/L	
QLN	Inquiry latch set ON	Q,A/B	
QLF	Inquiry latch set OFF	Q,A/B	
BQL	Branch if inquiry latch is ON	Q,A	
	CONSOLE TYPEWRITER OPERATIONS		
TYP	Type	A/L	
	ARITHMETIC OPERATIONS		
A#	Add to accumulator #	A/L	F
ZA#	Zero accumulator # and add	A/L	F
AA	Add absolute to accumulator 1	A/L	F
ZAA	Zero accumulator 1 and add absolute	A/L	F
AS#	Add to storage from accumulator #	A	F
AAS#	Add to absolute storage from accumulator #	A	F
S#	Subtract from accumulator #	A/L	F
ZS#	Zero accumulator # and subtract	A/L	F
SA	Subtract absolute from accumulator 1	A/L	F
ZSA	Zero accumulator 1 and subtract absolute	A/L	F
SS#	Subtract accumulator # from storage	A	F
M	Multiply	A/L	F
D	Divide	A/L	F
	SHIFTING OPERATIONS		
SR#	Shift right accumulator #	N	
SRR#	Shift right and round accumulator #	N	
SL#	Shift left accumulator #	N	
SLC#	Shift left and count accumulator #	X	
SR	Shift right coupled	N	
SRR	Shift right and round coupled	N	
SL	Shift left coupled	N	
SLC	Shift left and count coupled	X	
SRS	Shift right split	N(P)	
SLS	Shift left split	N(P)	
	INDEX WORD OPERATIONS		
XL	Index word load	X,A/L	
XLIN	Index word load and interchange	X,A/L	
XU	Index word unload	X,A	
XA	Index word add to indexing portion	X,A/N	
XZA	Index word zero and add to indexing portion	X,A/N	
XS	Index word subtract from indexing portion	X,A/N	
XZS	Index word zero and subtract from indexing portion	X,A/N	
XSN	Index word set nonindexing portion	X,A/N	
BLX	Branch and load location in index word	X,A	

Mnemonic	Operation	Operand	
BXM	Branch if index word is minus	X,A	
BXN	Branch if indexing portion in index word is nonzero	X,A	
BCX	Branch compared index word	X,A	
BIX	Branch incremented index word	X,A	
BDX	Branch decremented index word	X,A	
	SWITCH OPERATIONS		
BAS	Branch if alteration switch is ON	SN,A	
BES	Branch if electronic switch is ON	SW,A	
BSF	Branch if electronic switch is ON and set OFF if ON	SW,A	
BSN	Branch if electronic switch is ON and set ON if OFF	SW,A	
HMFV	Halt mode for field overflow	A/B	
HMSC	Halt mode for sign change	A/B	
SMFV	Sense mode for field overflow	A/B	
SMSC	Sense mode for sign change	A/B	
ESN	Electronic switch set ON	SW,A/B	
ESF	Electronic switch set OFF	SW,A/B	
	DATA MOVEMENT OPERATIONS		
RG	Record gather	X,A/L	
RS	Record scatter	X,A/L	
EAN	Edit alphameric to numerical	X,A/L	
ENA	Edit numerical to alphameric	X,A/L	
ENB	Edit numerical to alphameric with blank insertion	X,A/L	
ENS	Edit numerical to alphameric with sign control	X,A/L	
	LOGIC OPERATIONS		
BM#	Branch if minus in accumulator #	A	
BZ#	Branch if zero in accumulator #	A	
BV#	Branch if overflow in accumulator #	A	
BFV	Branch if field overflow	A	
BSC	Branch if sign change	A	
BH	Branch if high	A	
BE	Branch if equal	A	
BL	Branch if low	A	
BCB	Branch if channel is busy	C,A	
BAL	Branch if any stacking latch is ON	A	
C#	Compare accumulator # to storage	A/L	F
CA	Compare absolute in accumulator 1 to absolute in storage	A/L	F
CD	Compare storage to digit	A P ,D	
CSA	Compare sign to alpha	A	
CSP	Compare sign to plus	A	
CSM	Compare sign to minus	A	
	MISCELLANEOUS OPERATIONS		
B	Branch	A	
HB	Halt and branch	A	
HP	Halt and proceed	A/B	
LE	Lookup equal only	A/L	F
LL	Lookup lowest	A/L	F
LEH	Lookup equal or high	A/L	F
MSA	Make sign alpha	A	

Mnemonic	*Operation*	*Operand*	
MSP	Make sign plus	A	
MSM	Make sign minus	A	
NOP	No operation	A/B	
PC	Priority control	A/L	
PR	Priority release	A/B	
ST#	Store accumulator #	A	F
STD#	Store digits from accumulator # and ignore sign	A	F
ZST#	Zero storage and store accumulator #	A	F
	FLOATING POINT OPERATIONS		
FA	Floating add	A/L	
FZA	Floating zero and add	A/L	
FAA	Floating add absolute	A/L	
FAD	Floating add double precision	A/L	
FADS	Floating add double precision and suppress normalization	A/L	
FS	Floating subtract	A/L	
FSA	Floating subtract absolute	A/L	
FM	Floating multiply	A/L	
FD	Floating divide	A/L	
FDD	Floating divide double precision	A/L	
FR	Floating round	A/B	
FBV	Floating branch overflow	A	
FBU	Floating branch underflow	A	

Appendix E: Note on Optional Characters

In certain cases, special characters used on IBM printers and other equipment have optional equivalents. In each case the character must be punched according to the card code, regardless of which option has been chosen for printing on the printer in a given installation.

The special characters which have been used in this manual and their optional equivalents for each type wheel configuration available are given in the following table.

Character Used in This Manual	IBM Card Code	Type Wheel Configuration								
		A	B	C	D	E	F	G	H	K
(	0-8-4	%	%	%	%	%	(	%	(	(
) and ¤	12-8-4	¤	¤	¤	¤	<	)	¤	)	)
@	8-4	@	@	@	@	>	-	-	'	@
+	12	&	/	&	-	-	+	+	+	+
=	8-3	#	#	#	#	#	=	+	=	=
/	0-1	1	&	0	/	&	/	/	/	/

Appendix F: Glossary

ACTUAL ADDRESS

The word "actual" usually refers to machine language. An actual address is the same as an absolute or machine address.

ADDRESS ADJUSTMENT

Address adjustment refers to the procedure of changing an address at process time by means of an increment or decrement placed after the named address, i.e., NAME + 26.

ADDRESS CONSTANT (ADCON)

An adcon is a numerical literal created by entering ± ANYLABEL (a symbolic address written elsewhere in the program) in the operand of an instruction. The actual address assigned to ANYLABEL and the sign of the adcon entry become the address constant.

ALPHAMERIC

A term which refers to symbols that are numerical digits, alphabetic characters, or special characters.

ASSEMBLY PROGRAM

A translation program which substitutes machine-language instructions for symbolic instructions, assigns storage locations, and performs other activities necessary to produce the final object program.

7070/7074 AUTOCODER

Autocoder is a symbolic programming system consisting of a symbolic language and a processor; this system is designed for use in installations which have a minimum of six 729 Model IV (or Model II) tape units and a machine with 5,000 words of core storage. The language consists of symbolic machine instructions and generator-type macro-instructions; the processor converts a program written in this language to machine language.

7070/7074 BASIC AUTOCODER

Basic Autocoder is a symbolic programming system consisting of a symbolic language and a processor; this system is designed for use in installations which have a minimum of one IBM 7500 Card Reader, one IBM 7550 Card Punch, and a machine with 5,000 words of core storage. The language consists of symbolic machine instructions; the processor converts a program written in this language to machine language.

BRANCH CONSTANT

A branch constant is an instruction which is used to provide the address of a field to a subroutine; it is interrogated by the subroutine but it is never executed.

COLLATING SEQUENCE

The relative order of precedence which a computer assigns to the numbers, letters, and special characters for compare operations.

COMPILE

To produce a machine-language routine from a routine written in non-machine language. See also: COMPILER.

COMPILER

A compiler is a complex program in which several different functions are performed. It typically includes the following:

1. Extensive program analysis during which information is collected or tabulated for later recombination.
2. Generation of instructions by synthesis of tabulated information and use of skeleton or model routines.
3. Translation of symbolic instructions into machine language.

A compiler is itself a routine, not a machine—although a machine could be built to do compiling.

7070/7074 COMPILER SYSTEMS TAPE

This tape combines and links 7070/7074 Autocoder, 7070 FORTRAN, and the 7070/7074 Report Program Generator, permitting the use of these compilers with a minimum of effort and a maximum of efficiency.

CONDENSED CARD FORMAT

This is a format for placing several machine language instructions on a single card along with information sufficient to load the instructions into their proper storage locations.

CONTROL CARD

A card which contains the parameters required to set up a generalized program for one particular application.

DEBUGGING

The process of locating errors in a computer routine and correcting them.

EXPRESSION

An element of the source language where a combination of several names and operators may be used, as well as a single name or address.

FIELD DEFINER(S)

A number placed after an address to indicate the particular digit(s) in a word which are occupied by a field.

FIELD DEFINITION

Indication of the starting and ending positions of a field within a word, or the starting and ending positions of a part of a field relative to the field itself; for example, SYMBOL(6,9).

7070 FORTRAN

FORTRAN is a symbolic programming system consisting of a symbolic language and a processor. The language, which closely resembles the language of mathematics, is essentially problem-oriented rather than machine-oriented. The processor converts a program written in this language to 7070/7074 Autocoder language. The Autocoder program is converted to a machine-language object program for a 7070 or a 7074 by the Autocoder processor. Other IBM Data Processing Systems are equipped with processors which convert programs coded in the FORTRAN language to their respective machine-oriented languages.

7070/7074 FOUR-TAPE AUTOCODER

Four-Tape Autocoder is a symbolic programming system consisting of a symbolic language and a processor; this system is designed for use in installations which have a minimum of four 729 Model IV (or Model II) tape units and a machine with 5,000 words of core storage. The language consists of symbolic machine instructions and substitution-type macro-instructions; the processor converts a program written in this language to machine language.

GENERATOR

A program which accepts input parameters and uses them to modify skeleton instructions or skeleton routines to produce the desired output routine. A small number of parameters are capable of producing a large number of output instructions.

INDEXING

Refers to the procedure of changing an address, at object time, according to the contents of the indexing portion of a specified index word.

INSTRUCTION

An instruction is generally a single entry in symbolic machine language or in machine language, as opposed to a statement, or macro-instruction, which usually means a language entry that can produce many machine-language entries.

LITERAL

A literal is the actual data to be operated on by an instruction, as distinguished from the location or address of the data.

MACHINE LANGUAGE

A language in which the instructions are in a form which may be executed by the computer without translation.

MACRO GENERATOR

A macro generator, an abbreviation of macro-instruction generator, is that part of Autocoder which processes macro statement entries. It generates a sequence of symbolic machine instructions which, collectively, perform the operation specified by the macro-instruction entry. Each macro generator is associated with a given macro statement and processes all occurrences of that macro statement in a program.

MACRO-INSTRUCTION

A symbolically coded instruction resulting in a group of machine-language instructions which will perform a desired operation.

MNEMONIC

"Mnemonic" means "aiding memory." The term is used to describe operation codes which are written in a symbolic notation to make them easier to remember than the actual operation codes.

NUMERICAL

In the 7070 or the 7074, numerical refers to either a field with a plus or minus sign rather than an alpha sign or to data which is to be treated as numbers, whether in single-digit or double-digit form.

OBJECT MACHINE

The machine on which an object program is to be run. See: PROCESSOR MACHINE.

OBJECT PROGRAM

The output from a processor. In this case, a 7070/7074 machine-language program assembled from a source program coded in symbolic language.

OBJECT TIME

The time at which the object program is being run. This is opposed to process time, the time at which a compiler, such as Autocoder, is being run.

OFF-LINE

Operation of input/output and other devices not under direct computer control. Most commonly used to designate the transfer of information between magnetic tapes and other input/output media.

ON-LINE

Operation of an input/output device as a component of the computer, under programmed control.

OPERAND

The operand is the factor or data acted upon during an operation; it may be a result, a parameter, or an indication of the location of the next instruction. Also, the entire field beginning in column 21 of the Autocoder source card and coding sheet is considered to be the operand.

OPERATION CODE

The operation code designates the machine function to be performed. Distinction is sometimes made between a "mnemonic operation code," such as ZA1, and its equivalent "machine operation code," +13.

OPERATOR

The term operator usually refers to such characters as +, −, =, which are said to "operate" on quantities.

PARAMETER

A factor which is left unspecified and to which the user may assign a value.

PROCESS TIME

The time at which the source program is being changed into an object program by a compiler, such as Autocoder. This is opposed to object time, the time at which the object program is being run.

PROCESSOR

A program which performs the functions of assembly, compilation, generation, or any similar functions to convert a source program into the desired object program.

PROCESSOR MACHINE

The machine on which a processor is to be run.

PROGRAMMING SYSTEM

A programming system is any method of programming problems other than machine language, such as Autocoder. A system consists of a language and its associated processor(s).

REPORT

A printed document which presents data arranged in an orderly manner for ease of reference.

REPORT GENERATION

A technique for producing complete reports given the content and format of the input file and the desired content and format of the output reports.

7070/7074 REPORT PROGRAM GENERATOR

The 7070/7074 Report Program Generator consists of coding forms and a compiler. The format of the input file and the specifications for the output report are placed on the coding sheets which are then punched into cards and used as input for the compiler. The system produces a program in 7070/7074 Autocoder language which is converted to a machine-language object program for a 7070 or a 7074 by the Autocoder processor.

SORT

To place a file of records in order according to a specified sequence.

SOURCE LANGUAGE

The language in which a program is coded, e.g., the 7070/7074 Autocoder language.

SOURCE PROGRAM

The original coding of a program, usually coding in a language other than machine language.

SPECIAL CHARACTER

One of a set of special symbols. Some common special characters are:

$ + – * ( ) / , . ⌑ =

STATEMENT

Usually a source-language entry on the coding sheet, especially a line which might eventually produce several machine-language instructions, such as the ARITH statement or the GET statement.

STORAGE

Any medium into which data may be transferred and where it may be retained for later use.

SUBROUTINE

A subroutine is usually a series of instructions to perform some specific mathematical or logical operation. Subroutines are entered by a Branch instruction, as opposed to macro-instructions which are normally entered sequentially.

SYMBOL

In Autocoder, symbol is used to refer to a name used instead of a machine address. Thus "symbolic address," "symbolic name," or "symbolic label," conveys that one is not specifying machine addresses.

SYMBOLIC ADDRESS

An alphameric name used in place of an actual machine address.

SYMBOLIC LANGUAGE

A symbolic language is a collection of mnemonic symbols with rules of usage, such as Autocoder; the symbols are used in programming to represent operation codes, functions, and/or addresses. Symbolic-language coding must be translated into machine language to be used by the computer.

SYMBOLIC MACHINE LANGUAGE

Symbolic machine language is a language which is similar to machine language except for symbolic addresses and mnemonic operation codes. Symbolic machine instructions are sometimes referred to as "one-for-one" instructions since each instruction encountered in a source program will cause the corresponding machine language instruction to be inserted in the object program.

Appendix G: Illustration of Autocoder Programming

Input Statements

The listing on the following page contains the input statements of a sample payroll problem. The purpose of this problem is to illustrate coding with 7070/7074 Autocoder, including the use of input/output statements and macro-instructions.

DA statements define the formats of the input, output, and detail files, the file from which checks will be printed, and temporary storage fields. A DLINE statement describes the check format. An error message and a space for the insertion of an identification number are provided under a DC statement, for use if the desired record fails to appear in the master file. The processing routine follows the declarative statements in the listing.

Generated Coding

The listings which follow contain coding generated from the previously described input statements. This listing is incomplete; the DTF entries, DIOCS entries, and the major portion of the generated input/output instructions have been omitted.

```
AF01           *        AN ILLUSTRATION OF AUTOCODER PROGRAMMING
AF02           *
AF03           *        DESCRIPTION OF THE INPUT MASTER FILE
AF04     IMASTER      DA       10,RDW,0+IMASTERX
AF05     IMANNUMBER            00,09A10
AF06     INAME                 10,39'
AF07     IDEPENDNTS            40,41A2
AF07A    IPAYRATE              42,46A2.3
AF08     ICOMISRATE            47,49A.3
AF09     IYTDPAY               50,56A5.2
AF10     IYTDFICA              65,69A3.2
AF11           *
AF12           *        DESCRIPTION OF THE OUTPUT MASTER FILE
AF13     MASTEROUT    DA       10,RDW,0+OMASTERX
AF14                           00,69
AF15           *
AF16           *        DESCRIPTION OF THE DETAIL FILE
AF17     DETAIL       DA       20,RDW,0+DETAILX
AF18     DMANNUMBER            00,09A10
AF19     DHOURS                17,19A2.1
AF20     DSALES                22,29A6.2
AF21           *
AF22           *        DESCRIPTION OF THE CHECK FORMAT
AF23     CHECKLINE    DLINE
AF24     CMANNUMBER            2Z,ZZZ,ZZZ,ZZZ
AF25     CNAME                 18,32
AF26     CNETPAY               40$XX,XXZ.ZZ
AG01           *        DESCRIPTION OF THE FILE FROM WHICH CHECKS WILL BE PRINTED
AG02     CHECKTAPE    DA       10,RDW,0+CHECKX
AG03                           00,99
AG04           *
AG05           *        DESCRIPTION OF TEMPORARY STORAGE FIELDS
AG06     WORKAREA     DA       1
AG07     GROSSPAY              03,09A5.2
AG08     TAX                   13,19A5.2
AG09     FICA                  25,29A3.2
AG10     TFICA                 35,39A3.2
AG11     NETPAY                43,49A5.2
AG12           *
AG13     ERMESSAGE    DC       -RDW
AG14                           'MASTER IS MISSING FOR MANNUMBER'
AG15     ERRORNO               '          '
AG16           *
AG17           *    PROGRAM
AG18     IOPEN        OPEN     IMASTER,DETAIL,MASTEROUT,CHECKTAPE
AG19     START        GET      DETAIL
AG20     NEXTMASTER   GET      IMASTER
AG21                  COMP     DMANNUMBER,IMANNUMBER,NOMASTER,,NODETAIL
AG22                  ARITH    GROSSPAY=IPAYRATE*DHOURS+DSALES*ICOMISRATE
AG23                  ARITH    TAX=.18*(GROSSPAY-IDEPENDNTS*13.00)
AG24                  ZSIGN    TAX,,,ZEROTAX
AG25     FICATEST     ARITH    FICA=GROSSPAY*.03
AH01                  ARITH    TFICA=IYTDFICA+FICA-144.00
AH02                  ZSIGN    TFICA,FICALC,,FICALC
AH03                  ARITH    FICA=FICA-TFICA
AH04     FICALC       ARITH    IYTDFICA=IYTDFICA+FICA
AH05                  ARITH    IYTDPAY=IYTDPAY+GROSSPAY
AH06                  ARITH    NETPAY=GROSSPAY-TAX-FICA
AH07                  EDMOV    IMANNUMBER TO CMANNUMBER,INAME TO CNAME,NETPAY TO
AH08                           CNETPAY
AH08A          *    PREPARE TAPE RECORD FOR PRINTING CHECKS OFFLINE
AH09                  PUT      CHECKLINE IN CHECKTAPE
AH10                  PUTX     IMASTER IN MASTEROUT
AH11                  B        START
AH12     ZEROTAX      ZERO     TAX
AH13                  B        FICATEST
AH14     NOMASTER     MOVE     DMANNUMBER TO ERRORNO
AH15                  TYP      ERMESSAGE
AH16                  NOP
AH17                  B        START
AH18     NODETAIL     PUTX     IMASTER IN MASTEROUT
AH19                  B        NEXTMASTER
AH20     EOFDETAIL    BSN      1,IEND
AH21     RNOUTMASTR   PUTX     IMASTER IN MASTEROUT
AH22                  GET      IMASTER
AH23                  B        RNOUTMASTR
AH24     EOFMASTER    BSN      1,IEND
AH25     RNOUTDTAIL   MOVE     DMANNUMBER TO ERRORNO
AI01                  TYP      ERMESSAGE
AI02                  NOP
AI03                  GET      DETAIL
AI04                  B        RNOUTDTAIL
AI05     IEND         END
AI06     END          CNTRL    IOPEN
```

00084

```
PAGE AY    PROGRAM                                                                          PAGE AY
LN CDREF  LABEL       OP    OPERAND                                   CDNO FD  LOC  INSTRUCTION     REF

34 AF01    *        AN ILLUSTRATION OF AUTOCODER PROGRAMMING
35 AF02    *
36 AF03    *          DESCRIPTION OF THE INPUT MASTER FILE
37 AF04    IMASTER     DA    10,RDW,0+IMASTERX                                   +0013641443
38          X                                                          00219     1364 +0013741380   1364
39          X                                                                    1365 +0013811387   1365
40          X                                                                    1366 +0013881394   1366
41          X                                                                    1367 +0013951401   1367
42          X                                                                    1368 +0014021408   1368
43          X                                                          00220     1369 +0014091415   1369
44          X                                                                    1370 +0014161422   1370
45          X                                                                    1371 +0014231429   1371
46          X                                                                    1372 +0014301436   1372
47          X                                                                    1373 -0014371443   1373
48 AF05    IMANNUMBER        00,09A10                                       09  1374                0000
```

```
PAGE AZ    PROGRAM                                                                          PAGE AZ
LN CDREF  LABEL       OP    OPERAND                                   CDNO FD  LOC  INSTRUCTION     REF

01 AF06    INAME             10,39'                                         09  1375                0001
02 AF07    IDEPENDNTS        40,41A2                                        01  1378                0004
03 AF07A   IPAYRATE          42,46A2.3                                      26  1378                0004
04 AF08    ICOMISRATE        47,49A.3                                       79  1378                0004
05 AF09    IYTDPAY           50,56A5.2                                      06  1379                0005
06 AF10    IYTDFICA          65,69A3.2                                      59  1380                0006
07 AF11    *
08 AF12    *          DESCRIPTION OF THE OUTPUT MASTER FILE
09 AF13    MASTEROUT   DA    10,RDW,0+OMASTERX                                   +0014441523
10          X                                                          00221     1444 +0014541460   1444
11          X                                                                    1445 +0014611467   1445
12          X                                                                    1446 +0014681474   1446
13          X                                                                    1447 +0014751481   1447
14          X                                                                    1448 +0014821488   1448
15          X                                                          00222     1449 +0014891495   1449
16          X                                                                    1450 +0014961502   1450
17          X                                                                    1451 +0015031509   1451
18          X                                                                    1452 +0015101516   1452
19          X                                                                    1453 -0015171523   1453
20 AF14                      00,69                                          09  1454                0000
21 AF15    *
22 AF16    *          DESCRIPTION OF THE DETAIL FILE
23 AF17    DETAIL      DA    20,RDW,0+DETAILX                                    +0015241603
24          X                                                          00223     1524 +0015441546   1524
25          X                                                                    1525 +0015471549   1525
26          X                                                                    1526 +0015501552   1526
27          X                                                                    1527 +0015531555   1527
28          X                                                                    1528 +0015561558   1528
29          X                                                          00224     1529 +0015591561   1529
30          X                                                                    1530 +0015621564   1530
31          X                                                                    1531 +0015651567   1531
32          X                                                                    1532 +0015681570   1532
33          X                                                                    1533 +0015711573   1533
34          X                                                          00225     1534 +0015741576   1534
35          X                                                                    1535 +0015771579   1535
36          X                                                                    1536 +0015801582   1536
37          X                                                                    1537 +0015831585   1537
38          X                                                                    1538 +0015861588   1538
39          X                                                          00226     1539 +0015891591   1539
40          X                                                                    1540 +0015921594   1540
41          X                                                                    1541 +0015951597   1541
42          X                                                                    1542 +0015981600   1542
43          X                                                                    1543 -0016011603   1543
44 AF18    DMANNUMBER        00,09A10                                       09  1544                0000
45 AF19    DHOURS            17,19A2.1                                      79  1545                0001
46 AF20    DSALES            22,29A6.2                                      29  1546                0002
47 AF21    *
48 AF22    *          DESCRIPTION OF THE CHECK FORMAT
```

```
PAGE BA    PROGRAM                                                                                  PAGE BA

LN CDREF  LABEL        OP     OPERAND                                              CDNO  FD  LOC   INSTRUCTION     REF

01 AF23   CHECKLINE   DLINE                                                        00228               +0016041613
02 AF24   CMANNUMBER          ZZ,ZZZ,ZZZ,ZZZ                                              29  1604                  1604
03 AF25   CNAME               18,32                                                00229  49  1607                  1607
07 AF26   CNETPAY             40$XX,XXZ.ZZ                                                89  1611                  1611
08 AG01   *            DESCRIPTION OF THE FILE FROM WHICH CHECKS WILL BE PRINTED
09 AG02   CHECKTAPE   DA      10,RDW,0+CHECKX                                                          +0016141723
10       X                                                                         00230      1614 +0016241633     1614
11       X                                                                                    1615 +0016341643     1615
12       X                                                                                    1616 +0016441653     1616
13       X                                                                                    1617 +0016541663     1617
14       X                                                                                    1618 +0016641673     1618
15       X                                                                         00231      1619 +0016741683     1619
16       X                                                                                    1620 +0016841693     1620
17       X                                                                                    1621 +0016941703     1621
18       X                                                                                    1622 +0017041713     1622
19       X                                                                                    1623 -0017141723     1623
20 AG03                       00,99                                                       09  1624                  0000
21 AG04   *
22 AG05   *            DESCRIPTION OF TEMPORARY STORAGE FIELDS
23 AG06   WORKAREA    DA      1                                                                        +0017241728
24 AG07   GROSSPAY            03,09A5.2                                                   39  1724                  1724
25 AG08   TAX                 13,19A5.2                                                   39  1725                  1725
26 AG09   FICA                25,29A3.2                                                   59  1726                  1726
27 AG10   TFICA               35,39A3.2                                                   59  1727                  1727
28 AG11   NETPAY              43,49A5.2                                                   39  1728                  1728
29 AG12   *
30 AG13   ERMESSAGE   DC      -RDW                                                                     +0017291738
31       X                                                                         00232      1729 -0017301738     1729
32 AG14                       'MASTER IS MISSING FOR MANNUMBER'                    00233  09  1730 '7461828365     1730
33       X                                                                                09  1731 '7900698200     1731
34       X                                                                                09  1732 '7469828269     1732
35       X                                                                                09  1733 '7567006676     1733
36       X                                                                                09  1734 '7900746175     1734
37       X                                                                         00234  09  1735 '7584746265     1735
38       X                                                                                01  1736 '79             1736
39 AG15   ERRORNO                                                                         29  1736 '  00000000     1736
40       X                                                                                09  1737 '0000000000     1737
41       X                                                                                01  1738 '00             1738
42 AG16   *
43 AG17   *        PROGRAM
44 AG18   IOPEN       OPEN    IMASTER,DETAIL,MASTEROUT,CHECKTAPE
45       X IOPEN      BLX     IOCSIXG,IOC.IOPEN                                    00235      1739 +0200040578
46       X            B       TAPEFILEIM                                                      1740 +0100091327
47       X            B       TAPEFILEDI                                                      1741 +0100091345
48       X            B       TAPEFILEMO                                                      1742 +0100091336
```

```
CDREF   LABEL         OP     OPERAND                                          CDNO  FD  LOC  INSTRUCTION   REF
          X             B      TAPEFILECO                                                  1743 +0100091354
          X             NOP                                                     00236      1744 -0100000000
AG19      START         GET    DETAIL
          X START       BIX    DETAILB,*+2                                                 1745 +4900051747
          X             BLX    DETAILX,IOC.NSE03A                                          1746 +0200062039
          X             XL     DETAILX,0+DETAILB                                           1747 +4505060000
AG20      NEXTMASTER    GET    IMASTER
          X NEXTMASTER  BIX    IMASTERB,*+2                                                1748 +4900071750
          X             BLX    IMASTERX,IOC.NSE01A                              00237      1749 +0200081945
          X             XL     IMASTERX,0+IMASTERB                                         1750 +4507080000
AG21                    COMP   DMANNUMBER,IMANNUMBER,NOMASTER,,NODETAIL
          X             ZA2    DMANNUMBER(0,9)+DETAILX                                     1751 +2306090000
          X             S2     IMANNUMBER(0,9)+IMASTERX                                    1752 -2408090000
          X             BV2    *+2                                                         1753 +2100091755
          X             BZ2    *+3                                              00238      1754 +2000091757
          X             BM2    NOMASTER                                                    1755 -2000091881
          X             B      NODETAIL                                                    1756 +0100091891
AG22                    ARITH  GROSSPAY=IPAYRATE*DHOURS+DSALES*ICOMISRATE                                   M
          X             ZA3    IPAYRATE(0,4)+IMASTERX                                      1757 +3308260004
          X             M      DHOURS(0,2)+DETAILX                                         1758 +5306790001
          X             SR     1                                                00239      1759 -5000000001
          X             ST2    COMAREA.A(0,9)+1                                            1760 +2200092122
          X             ZA3    DSALES(0,3)+DETAILX                                         1761 +3306250002
          X             M      ICOMISRATE(0,2)+IMASTERX                                    1762 +5308790004
          X             A2     COMAREA.A(0,9)+1                                            1763 +2400092122
          X             SRR2   1                                                00240      1764 +5000002101
          X             ST2    GROSSPAY(0,6)                                               1765 +2200391724
AG23                    ARITH  TAX=.18*(GROSSPAY-IDEPENDNTS*13.00)                                          M
          X             ZS3    IDEPENDNTS(0,3)+IMASTERX                                    1766 -3308030004
          X             M      +1300                                                       1767 +5300142124
          X             A2     GROSSPAY(0,6)                                               1768 +2400391724
          X             ZA3    +18                                              00241      1769 +3300232125
          X             M      9992                                                        1770 +5300099992
          X             SRR    2                                                           1771 -5000000102
          X             ST2    TAX(0,6)                                                    1772 +2200391725
AG24                    ZSIGN  TAX,,,ZEROTAX
          X             ZA1    TAX(0,6)                                                    1773 +1300391725
          X             BZ1    M.24                                             00242      1774 +1000091777
          X             CSM    TAX                                                         1775 -0300601725
          X             BE     ZEROTAX                                                     1776 -4100091878
          X M.24        NOP                                                                1777 -0100000000
          X ORIGIN      CNTRL  *-1
AG25      FICATEST      ARITH  FICA=GROSSPAY*.03
          X FICATEST    ZA3    GROSSPAY(0,6)                                               1777 +3300391724
          X             M      +03                                                         1778 +5300012125
          X             SRR    2                                                00243      1779 -5000000102
          X             ST2    FICA(0,4)                                                   1780 +2200591726
AH01                    ARITH  TFICA=IYTDFICA+FICA-144.00                                                   M
```

```
LN CDREF  LABEL     OP     OPERAND                                               CDNO FD  LOC  INSTRUCTION   REF
             X          ZA2    FICA(0,3)                                                      1781 +2300581726
             X          SL2    2                                                              1782 +5000002202
             X          A2     IYTDFICA(0,4)+IMASTERX                                         1783 +2408590006
             X          S2     +14400                                                00244    1784 -2400592124
             X          ST2    TFICA(0,4)                                                     1785 +2200591727
   AH02                 ZSIGN  TFICA,FICALC,,FICALC
             X          ZA1    TFICA(0,4)                                                     1786 +1300591727
             X          BZ1    FICALC                                                         1787 +1000091794
             X          CSM    TFICA                                                          1788 -0300601727
             X          BE     FICALC                                                00245    1789 -4100091794
   AH03                 ARITH  FICA=FICA-TFICA                                                                   M
             X          ZS2    TFICA(0,3)                                                     1790 -2300581727
             X          SL2    2                                                              1791 +5000002202
             X          A2     FICA(0,4)                                                      1792 +2400591726
             X          ST2    FICA(0,4)                                                      1793 +2200591726
   AH04      FICALC     ARITH  IYTDFICA=IYTDFICA+FICA                                                            M
             X FICALC   ZA2    FICA(0,3)                                             00246    1794 +2300581726
             X          SL2    2                                                              1795 +5000002202
             X          A2     IYTDFICA(0,4)+IMASTERX                                         1796 +2408590006
             X          ST2    IYTDFICA(0,4)+IMASTERX                                         1797 +2208590006
   AH05                 ARITH  IYTDPAY=IYTDPAY+GROSSPAY                                                          M
             X          ZA2    GROSSPAY(0,3)                                                  1798 +2300361724
             X          SL2    2                                                     00247    1799 +5000002202
             X          A2     IYTDPAY(0,6)+IMASTERX                                          1800 +2408060005
             X          ST2    IYTDPAY(0,6)+IMASTERX                                          1801 +2208060005
   AH06                 ARITH  NETPAY=GROSSPAY-TAX-FICA                                                          M
             X          ZS2    TAX(0,3)                                                       1802 -2300361725
             X          S2     FICA(0,3)                                                      1803 -2400581726
             X          SL2    2                                                     00248    1804 +5000002202
             X          A2     GROSSPAY(0,6)                                                  1805 +2400391724
             X          ST2    NETPAY(0,6)                                                    1806 +2200391728
   AH07                 EDMOV  IMANNUMBER TO CMANNUMBER,INAME TO CNAME,NETPAY TO
   AH08                        CNETPAY
             X          ZA2    IMANNUMBER(0,9)+IMASTERX                                       1807 +2308090000
             X          ST2    COMAREA.A+2                                                    1808 +2200092123
             X          XZA    MACREG.01,COMAREA.A+2                                 00249    1809 +4600092123
             X          ENA    MACREG.01,EDMOV02.A                                            1810 +5600092111
             X          ZA3    COMAREA.A(0,1)                                                 1811 +3300012121
             X          ST3    CMANNUMBER(0,1)                                                1812 +3200231604
             X          ZA3    ',' 
                                                                                              1813 +3300452127
             X          ST3    CMANNUMBER(2,3)                                       00250    1814 +3200451604
             X          ZA3    COMAREA.A(2,5)                                                 1815 +3300252121
             X          ST3    CMANNUMBER(4,7)                                                1816 +3200691604
             X          ZA3    COMAREA.A(6,7)                                                 1817 +3300672121
             X          ST3    CMANNUMBER(8,9)                                                1818 +3200011605
             X          ZA3    ','                                                   00251    1819 +3300452127
             X          ST3    CMANNUMBER(10,11)                                              1820 +3200231605
             X          ZA3    COMAREA.A(8,9)                                                 1821 +3300892121
```

```
PAGE BD     PROGRAM                                                                                         PAGE BI

CDREF  LABEL    OP     OPERAND                                      CDNO FD   LOC  INSTRUCTION      REF

   X            ST3    CMANNUMBER(12,13)                                      1822 +3200451605
   X            ZA3    COMAREA.A(10,13)                                       1823 +3300032122
   X            ST3    CMANNUMBER(14,17)                            00252     1824 +3200691605
   X            ZA3    ','                                                    1825 +3300452127
   X            ST3    CMANNUMBER(18,19)                                      1826 +3200011606
   X            ZA3    COMAREA.A(14,19)                                       1827 +3300492122
   X            ST3    CMANNUMBER(20,25)                                      1828 +3200271606
   X            ZA1    INAME(0,9)+IMASTERX                          00253     1829 +1308090001
   X            ZA2    INAME(10,19)+IMASTERX                                  1830 +2308090002
   X            ST2    CNAME(16,19)                                           1831 +2200031609
   X            SR     4                                                      1832 -5000000004
   X            ST2    CNAME(6,15)                                            1833 +2200091608
   X            ST1    CNAME(0,5)                                   00254     1834 +1200491607
   X            ZA2    INAME(20,29)+IMASTERX                                  1835 +2308090003
   X            ST2    CNAME(26,29)                                           1836 +2200031610
   X            SR2    4                                                      1837 +5000002004
   X            ST2    CNAME(20,25)                                           1838 +2200491609
   X            ZA2    NETPAY(0,6)                                  00255     1839 +2300391728
   X            ST2    COMAREA.A+2                                            1840 +2200092123
   X            XZA    MACREG.01,COMAREA.A+2                                  1841 +4600092123
   X            ENA    MACREG.01,EDMOV02.A                                    1842 +5600092111
   X            ZA3    '     '                                                1843 +3300092126
   X            ST3    CNETPAY(0,1)                                 00256     1844 +3200891611
   X            ST3    CNETPAY(2,9)                                           1845 +3200071612
   X            SLC2   MACREG.02                                              1846 +5000102300
   X            ZA3    '$'                                                    1847 +3300232127
   X            B      M.22-3+MACREG.02                                       1848 +0110092109
   X   M.23     ST3    CNETPAY(0,1)                                 00257     1849 +3200891611
   X            ZA3    COMAREA.A(6,7)                                         1850 +3300672121
   X            ST3    CNETPAY(2,3)                                           1851 +3200011612
   X            ZA3    COMAREA.A(8,9)                                         1852 +3300892121
   X            ST3    CNETPAY(4,5)                                           1853 +3200231612
   X            ZA3    ','                                          00258     1854 +3300452127
   X            ST3    CNETPAY(6,7)                                           1855 +3200451612
   X            ZA3    COMAREA.A(10,11)                                       1856 +3300012122
   X            ST3    CNETPAY(8,9)                                           1857 +3200671612
   X            ZA3    COMAREA.A(12,13)                                       1858 +3300232122
   X            ST3    CNETPAY(10,11)                               00259     1859 +3200891612
   X            ZA3    COMAREA.A(14,15)                                       1860 +3300452122
   X            ST3    CNETPAY(12,13)                                         1861 +3200011613
   X            ZA3    '.'                                                    1862 +3300012127
   X            ST3    CNETPAY(14,15)                                         1863 +3200231613
   X            ZA3    COMAREA.A(16,17)                             00260     1864 +3300672122
   X            ST3    CNETPAY(16,17)                                         1865 +3200451613
   X            ZA3    COMAREA.A(18,19)                                       1866 +3300892122
   X            ST3    CNETPAY(18,19)                                         1867 +3200671613
AH08A  *      PREPARE TAPE RECORD FOR PRINTING CHECKS OFFLINE
AH09            PUT    CHECKLINE IN CHECKTAPE
```

```
CDREF  LABEL        OP     OPERAND                         CDNO FD   LOC  INSTRUCTION   REF

           X          BIX    CHECKB,*+2                                1868 +4900111870
           X          BLX    CHECKX,IOC.NSE04A               00261     1869 +0200122086
           X          XL     CHECKX,0+CHECKB                           1870 +4511120000
           X          RG     CHECKX,IOC.PUT001                         1871 -6500122120
      AH10            PUTX   IMASTER IN MASTEROUT
           X          BIX    OMASTERB,*+2                              1872 +4900131874
           X          BLX    OMASTERX,IOC.NSE02A                       1873 +0200141992
           X          XL     OMASTERX,0+OMASTERB             00262     1874 +4513140000
           X          XU     IMASTERX,0+OMASTERB                       1875 -4513080000
           X          XU     OMASTERX,0+IMASTERB                       1876 -4507140000
      AH11            B      START                                     1877 +0100091745
      AH12 ZEROTAX    ZERO   TAX
           X ZEROTAX  ZA2    +0                                        1878 +2300002124
           X          STD2   TAX(0,6)                        00263     1879 -2200391725
      AH13            B      FICATEST                                  1880 +0100091777
      AH14 NOMASTER   MOVE   DMANNUMBER TO ERRORNO
           X NOMASTER ZA2    DMANNUMBER(0,9)+DETAILX                   1881 +2306090000
           X          ST2    ERRORNO(8,9)                              1882 +2200011737
           X          SR2    2                                         1883 +5000002002
           X          ST2    ERRORNO(0,7)                    00264     1884 +2200291736
           X          SL     20                                        1885 -5000000220
           X          ST2    ERRORNO(10,17)                            1886 +2200291737
           X          ST2    ERRORNO(18,19)                            1887 +2200011738
      AH15            TYP    ERMESSAGE                                 1888 +6900041729
      AH16            NOP                                    00265     1889 -0100090000
      AH17            B      START                                     1890 +0100091745
      AH18 NODETAIL   PUTX   IMASTER IN MASTEROUT
           X NODETAIL BIX    OMASTERB,*+2                              1891 +4900131893
           X          BLX    OMASTERX,IOC.NSE02A                       1892 +0200141992
           X          XL     OMASTERX,0+OMASTERB                       1893 +4513140000
           X          XU     IMASTERX,0+OMASTERB             00266     1894 -4513080000
           X          XU     OMASTERX,0+IMASTERB                       1895 -4507140000
      AH19            B      NEXTMASTER                                1896 +0100091748
      AH20 EOFDETAIL  BSN    1,IEND                                    1897 +6100301921
      AH21 RNOUTMASTR PUTX   IMASTER IN MASTEROUT
           X RNOUTMASTR BIX  OMASTERB,*+2                              1898 +4900131900
           X          BLX    OMASTERX,IOC.NSE02A             00267     1899 +0200141992
           X          XL     OMASTERX,0+OMASTERB                       1900 +4513140000
           X          XU     IMASTERX,0+OMASTERB                       1901 -4513080000
           X          XU     OMASTERX,0+IMASTERB                       1902 -4507140000
      AH22            GET    IMASTER
           X          BIX    IMASTERB,*+2                              1903 +4900071905
           X          BLX    IMASTERX,IOC.NSE01A             00268     1904 +0200081945
           X          XL     IMASTERX,0+IMASTERB                       1905 +4507080000
      AH23            B      RNOUTMASTR                                1906 +0100091898
      AH24 EOFMASTER  BSN    1,IEND                                    1907 +6100301921
      AH25 RNOUTDTAIL MOVE   DMANNUMBER TO ERRORNO
           X RNOUTDTAIL ZA2  DMANNUMBER(0,9)+DETAILX                   1908 +2306090000
```

```
PAGE BF     PROGRAM                                                                                         PAGE BF

LN CDREF  LABEL      OP    OPERAND                                          CDNO FD  LOC  INSTRUCTION     REF

01       X           ST2   ERRORNO(8,9)                                     00269    1909 +2200011737
02       X           SR2   2                                                         1910 +5000002002
03       X           ST2   ERRORNO(0,7)                                              1911 +2200291736
04       X           SL    20                                                        1912 -5000000220
05       X           ST2   ERRORNO(10,17)                                            1913 +2200291737
06       X           ST2   ERRORNO(18,19)                                   00270    1914 +2200011738
07 A101              TYP   ERMESSAGE                                                 1915 +6900041729
08 A102              NOP                                                             1916 -0100090000
09 A103              GET   DETAIL
10       X           BIX   DETAILB,*+2                                               1917 +4900051919
11       X           BLX   DETAILX,IOC.NSE03A                                        1918 +0200062039
12       X           XL    DETAILX,0+DETAILB                                00271    1919 +4505060000
13 A104              B     RNOUTDTAIL                                                1920 +0100091908
14 A105    IEND      END
15       X IEND      BLX   IOCSIXG,IOC.IEND                                          1921 +0200041227
16       X           NOP   0
```

Appendix H: Index of Messages

NOTE. For ARITH messages, which begin with an N, W or X followed by a two-digit number, see page 133.

Index

When more than one page reference for a particular subject is listed, the page number in *italics* indicates the major reference.

C28-6121-0

Printed in U.S.A. C28-6121-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York